University of Alberta

Numerical Solutions of Free-Surface Incompressible Flows

by

Boyan Bejanov ©

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Applied Mathematics

Department of Mathematical and Statistical Sciences

Edmonton, Alberta
Fall 2007

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-32915-3
Our file Notre référence
ISBN: 978-0-494-32915-3

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

What does not destroy me, makes me stronger.

And if you gaze for long into an abyss,
the abyss gazes also into you.

Friedrich Nietzsche

Abstract

This study is dedicated to the development, implementation, and analysis of a numerical method for computer simulations of multicomponent flows involving capillary free surfaces. For this purpose, a numerical scheme for the incompressible Navier-Stokes equations is presented. The scheme is an implementation of the projection method by the method of Finite Elements. The convection-diffusion sub-step is solved using a conforming linear finite element for the velocity, while the projection sub-step is solved using a nonconforming linear finite element for the velocity and piecewise constant pressure. The end-of-step velocity is locally pointwise divergence-free, which is a desired feature, since it allows for improved mass conservation.

This projection scheme is employed for computer simulations of multicomponent incompressible flows. Discontinuous pressure and low-order velocity approximations provide consistent handling of discontinuities in the solution, as long as computational cells are not intersected by moving interfaces. A robust algorithm for local grid alignment is proposed. A reference grid is maintained and used on every timestep to produce a new computational grid. Few nodes in the reference grid that are close to the interface are projected onto it, so that the computational grid contains no edges intersected by the interface and has the same connectivity as the reference grid. The unchanging connectivity makes parallelization easier and more effective. The interfaces are approximated in the vicinity of each node by a part of a sphere, which is also used for the computation of surface tension.

Both the proposed projection scheme and the local grid alignment are validated on a number of numerical examples.

To Maria, Yulia,
Nina, Krasimir and Veselka.

Table of Contents

Introduction **1**

1 Solving incompressible flows **2**
1.1 Introduction 2
1.2 Formulation of the problem and notation 5
1.3 Discretization 6
1.4 Error analysis 11
1.5 Implementation in two spatial dimensions 15
1.6 Implementation in three spatial dimensions 36
1.7 Conclusion 51

2 Discretization of problems with free capillary interfaces **53**
2.1 Introduction 53
2.2 Formulation of the problem and notation 56
2.3 Discretization 59
2.4 Grid alignment in two spatial dimensions 63
2.5 Grid alignment in three spatial dimensions 76
2.6 Conclusion 86

Conclusion **88**

References **90**

List of Tables

1 Comparison of the implementations of the projection step in 2d. 29
2 Comparison with $iso - \mathbb{P}_2\mathbb{P}_1$ element. 31
3 Convergence on diagonal grid. 33
4 Comparison of the implementations of the projection step in 3d. 50
5 Convergence of exact and decoupled projections. 50

List of Figures

1 Reference triangular element. 16
2 Triangular element with div-free basis. 24
3 Divergence-free basis function in 2d. 26
4 Sample 4×4 grid. 28
5 Error in velocity in $\ell^2(L^2)$-norm. 30
6 Error in velocity in $\ell^2(L^2)$-norm and $\ell^2(H^1)$-norm. 30
7 Convergence in space and in time for incremental scheme. . . . 32
8 Convergence in time for second order scheme. 32
9 Sample 4×4 diagonal grid. 33
10 Convergence of the pressure. 34
11 Lid-driven cafity flow. 35
12 Divergence-free basis function in 3d. 40
13 Sample $2\times2\times2$ grid. 49
14 Projecting node onto the interface. 65
15 2d element with three nodes on Γ. 66
16 Construction of the projection onto the interface. 69
17 2d lid-driven cavity with moving interface. 71
18 Convergence in 2d with moving interface. 72
19 Relaxation of 2d elliptical particle. 73
20 Relaxation of 2d star-shaped particle. 74
21 Simulation of the motion of 2d gas bubble. 75
22 Experimental result for the motion of 2d gas bubble. 76
23 Tetrahedral elements of poor quality. 77
24 Relaxation of an ellipsoidal particle. 84
25 The frequency of an oscillating bubble. 85
26 The shape of a rising bubble. 86

Introduction

The Navier-Stokes equations make arguably the most successful mathematical model. Governing the motion of fluids (liquids and gases), this model has been applied successfully to describe a wide variety of processes and phenomena, such as flows in a pipe and around an airfoil, ocean currents, the weather, and even the motion of stars in a galaxy. The Navier-Stokes equations are used for the design of aircrafts, naval vessels, automobiles, as well as the study of blood flow, climate change, the effects of pollution, etc. These few examples are barely touching the tip of the iceberg, and it is not hard to see why the study of analytic and approximate solutions of the Navier-Stokes equations is of great theoretical and practical importance.

The work presented in this volume is an investigation of the numerical solutions of the incompressible Navier-Stokes equations and their applications to computer simulations of flows involving capillary surfaces. Our ultimate goal is to extend and modify the existing techniques, as well as to develop a new numerical method that is suitable for the large scale simulations of the motion of a large number of bubbles immersed in another fluid.

The numerical solution of the incompressible Navier-Stokes equations presents a major problem in today's Computational Fluid Dynamics. This subject has attracted the attention of scientists for decades, but the need for the most accurate, fast, robust, and overall efficient solution method is yet to be satisfied. The greatest difficulty stems from the fact that pressure in an incompressible flow is not a dynamic variable; rather, it plays the role of a Lagrange multiplier, which keeps the velocity divergence-free. The resulting saddle-point problem can be avoided with the use of splitting schemes, where the pressure and the velocity are decoupled and solved for separately. In the first part of our investigation we develop, analyze, and implement an innovative splitting method that is specifically geared to meet our needs. Our main requirements are the use of low order velocity and discontinuous pressure approximations, as well as local mass conservation, both of which are desirable features of a method used in simulations of multiphase flows.

The second part of our study is dedicated to the application of the developed algorithm to simulations of flows involving two immiscible fluids separated by a sharp interface. These flows appear in various engineering applications, such as the design of chemical reactors, oil pipelines, combustion, hydrogen fuel cells, and others. In the solution methodology developed here, the computational mesh is aligned with the moving interface on every timestep without changing the connectivity, while the approximation of the interface is smoothed locally to facilitate accurate computations of discontinuous and singular quantities involved.

1 Solving incompressible flows

1.1 Introduction

The numerical solution of the Navier-Stokes equations, which govern the incompressible flow of viscous Newtonian fluids, involves two major difficulties. The first one is related to the stable approximation of the nonlinear advection term and is not of a major importance to the problems considered in this study, because the flows considered here involve relatively small Reynolds numbers. The second problem is related to the imposition of the incompressibility constraint. It is well known that the variational formulation of the Stokes problem yields a saddle-point problem, which is among the most difficult and computationally demanding problems to solve. The standard method for solving saddle point problems is the Uzawa iteration. It produces a highly accurate "exact" solution of the discrete equations, but it is rather expensive computationally since it involves two nested iterations. Thus, while being great for producing benchmark solutions of stationary problems, the Uzawa iteration is impractical for large scale simulations of time-dependent flows, where the saddle point problem must be solved once on every time step. A relatively cheaper alternative is provided by the so-called projection methods. They split the indefinite saddle point problem into two positive definite elliptic problems, which are easier to solve. The price paid for doing this is an additional splitting error, which cannot be avoided due to the full coupling of the unknown pressure and velocity in the original system.

One way to decouple the problem is through a predictor-corrector style time-marching algorithm. For example, we can proceed as follows:

- solve the momentum equation for the predictor velocity ignoring the pressure and the incompressibility constraint; then
- enforce the incompressibility condition by projecting the predictor velocity onto some divergence-free space, producing corrected velocity and pressure.

Such schemes fall into the class of pressure-correction projection methods, which are the most popular schemes, and arguably the fastest solution methods, for the unsteady Stokes problem available at present. Starting in the late 1960s with the work of Chorin and Temam [12, 80], projection methods

[0] A version of this chapter has been publshed.
B. Bejanov, J.-L. Guermond, and P.D. Minev. A locally DIV-free projection scheme for incompressible flows based on non-conforming elements. *Int. J. Numer. Meth. Fluids*, 49:549–568, 2005

have been developed, analyzed and tested for over three decades. An overview and recent developments can be found in [35, 39, 36]. The projection step is usually written as a Poisson problem for the pressure in which homogeneous Neumann boundary conditions are imposed. These inexact boundary conditions create a boundary layer of inaccuracy, which ultimately prevents the pressure from attaining full order of convergence in time [64]. The original Chorin-Temam algorithm is only first order accurate in time for the velocity, and the error in the pressure is asymptotically of order $O(\Delta t^{1/2})$. Second order in time schemes have been suggested subsequently in [32, 90]. In these schemes the pressure is treated explicitly in the first, advection-diffusion substep, and then corrected at the second, projection substep. These so-called incremental pressure-correction schemes achieve second order accuracy in time for the velocity, but the pressure is again plagued by spurious boundary layers around the prescribed boundaries and cannot achieve higher than $O(\Delta t)$ accuracy (with Dirichlet boundary conditions). They have been rigorously analyzed in [22, 72, 34]. In an attempt to remedy the situation, a slight modification in the algorithm was proposed in [86]. The adjusted methodology is now referred to as rotational incremental pressure-correction schemes. See also [40, 37], as well as [35], where it is shown that a scheme introduced in [47] is equivalent to a rotational pressure-correction scheme after an appropriate change of variables.

The dual approach to pressure-correction are the velocity-correction methods [46, 37, 39]. Here, the diffusion term is treated explicitly in the first step, which is now the projection step and includes the incompressibility constraint, while the second step accounts for viscous effects and corrects the velocity. The asymptotic behavior of these methods is equivalent to that of pressure-correction methods; they even suffer from a similar boundary layer. Again the rotational form improves the order of convergence in time. So far, it appears that a full second order in time for both the pressure and the velocity is achieved only by the so-called consistent splitting scheme [38]. In [35], it is shown that this scheme is equivalent, up to a suitable change of variables, to the so-called gauge method, proposed in [23]. However, this conjecture has not been proved rigorously yet and is only supported by numerical results on model problems.

When implementing a projection scheme using the Finite Element Method, the choice of elements is restricted by the so-called *inf-sup* condition which is a sufficient and necessary condition for the existence and uniqueness of the solution of the underlying stationary Stokes problem. It was derived independently by Ladyženskaja [48], Babuška [2, 3] and Brezzi [6]. Any discretization of the Stokes problem should satisfy the discrete version of the inf-sup condition in order to guarantee the uniqueness of the discrete solution. If the condition is not satisfied, the pressure may develop spurious modes (non-constant pres-

sures with discrete gradient 0), which leads to instability and generally to loss of accuracy. The choice of element also affects the way in which the incompressibility constraint is imposed.

There is a large number of two-dimensional and fewer, but also many, three-dimensional finite elements available to choose from. The simplest choice is the $\mathbb{P}_1\mathbb{P}_0$ triangular element, where the velocity is linear and the pressure is piecewise constant. This element is not inf-sup stable and is known to give zero velocity on diagonal grids [61]. The quadrilateral counterpart is the $\mathbb{Q}_1\mathbb{Q}_0$, which is also unstable. A stable element with linear velocity is the MINI $\mathbb{P}_1^+\mathbb{P}_1$ element with linear pressure. The velocity field is enriched by a cubic bubble function in each element, which provides stability. Another way in which stability is achieved is through clustering. The so-called $iso-\mathbb{P}_2\mathbb{P}_1$ element (also known as $\mathbb{P}_1\mathbb{P}_1$ macro element or Bercovier-Pironneau element) features linear velocity and linear pressure, although each element for the pressure is divided into four elements for the velocity. Both the MINI and the macro elements are available as 3-D tetrahedral elements, which are also stable. A popular class of second order elements are the $\mathbb{P}_2\mathbb{P}_1$ triangle/tetrahedron and the $\mathbb{Q}_2\mathbb{Q}_1$ quadrilateral/hexahedron, known as Taylor-Hood elements [79, 45]. These are inf-sup stable elements and, by virtue of their simplicity, they are very widely used.

The elements mentioned so far have a continuous pressure approximation; as a result, the mass is not conserved elementwise. An approximation for the pressure, which is discontinuous across element boundaries, allows for a per-element and even pointwise mass conservation. Examples of stable elements with discontinuous pressure are the $\mathbb{Q}_2\mathbb{P}_{-1}$ element introduced in [67] and the rotated multi-linear element with velocity spanned by $(x^2 - y^2, x, y, 1)$ and piecewise constant pressure proposed in [88]. Here we should also mention the Crouzeix-Raviart elements [15], some of which are nonconforming (with discontinuous velocity), although the flux through element faces is kept continuous. The Crouzeix-Raviart elements are considered by many to be impractical and of theoretical interest only due to their large number of degrees of freedom. They can be viewed as a particular case of the so-called discontinuous Galerkin (DG) methods, which, in the case of the Stokes problem, use fully discontinuous approximations for both the pressure and the velocity. An algorithm for the Navier-Stokes equations based on a DG approach was recently proposed in [13]; it revealed that a locally mass conservative approximation for the velocity can be used to construct a conforming and pointwise divergence-free approximation for it.

The main purpose of this study is the development and the implementation of a numerical technique suitable for large scale simulations of multi-component flows involving capillary free boundaries. One effect of the presence

of surface tension is the appearance of jump discontinuities in the pressure and the first spatial derivatives of the velocity. In order to approximate these jumps adequately, it is convenient to use an element with the lowest possible order of approximation for the velocity and a discontinuous pressure.

We have developed a version of the pressure-correction projection method that meets these requirements. It is based partly on $\mathbb{P}_1$ and partly on the linear nonconforming Crouzeix-Raviart elements (CR) for the velocity and $\mathbb{P}_0$ pressure. The $\mathbb{P}_1$ element is used to solve the momentum equation, avoiding the large linear system resulting from the CR element. The $\mathbb{P}_1\mathbb{P}_0$ element is inf-sup unstable and is known to produce only a trivial solution for the velocity on some grids (with homogeneous Dirichlet boundary conditions). For this reason we use the stable CR element in the projection step.

This chapter is organized as follows. In section 1.2 we formulate the incompressible Navier-Stokes problem and introduce notation. In section 1.3 we provide a detailed overview of the two finite elements and describe the projection method used. Section 1.4 is dedicated to the analysis of the proposed algorithm. Sections 1.5 and 1.6 discuss various issues of implementing the scheme in 2d and 3d respectively, concluding with numerical illustrations. All findings presented in this chapter are summarized in section 1.7.

1.2 Formulation of the problem and notation

We consider a viscous Newtonian fluid occupying an open, bounded, and connected N-dimensional (N=2 or 3) domain Ω and undergoing incompressible flow. After proper nondimensionalization we formulate the following Dirichlet boundary problem for the incompressible Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\,\mathbf{u} - \frac{1}{Re}\nabla^2\mathbf{u} + \nabla p = \mathbf{f} \qquad \text{in } \Omega\times(0,T), \tag{1.1}$$

$$\nabla\cdot\mathbf{u} = 0 \qquad \text{in } \Omega\times(0,T), \tag{1.2}$$

$$\mathbf{u}|_{\partial\Omega} = \mathbf{u}_{bc} \qquad \text{for } t\in(0,T), \tag{1.3}$$

$$\mathbf{u}|_{t=0} = \mathbf{u}_0 \qquad \text{in } \Omega. \tag{1.4}$$

Here $\mathbf{u}$ and p are the unknown velocity and pressure, $\mathbf{f}$ is an external force. The Reynolds number Re is defined as

$$Re = \frac{\rho u_c l_c}{\mu}, \tag{1.5}$$

where ρ and μ are the density and the dynamic viscosity of the fluid, while u_c and l_c are some appropriately chosen characteristic speed and length.

We use standard notation for functional spaces. L^2 denotes the space of

all functions that are square integrable in Ω, and H^m is the Sobolev space of all functions in L^2 having derivatives of order up to and including m that are also in L^2. We use $(\cdot,\cdot)_m$ and $\|\cdot\|_m$ to denote the inner product and the norm in H^m. When the subscript is omitted, we mean the inner product or norm in L^2 (which is the same as m=0). We also denote

$$H_0^m = \{\phi \in H^m |\ \phi|_{\partial\Omega} = 0\}\,. \tag{1.6}$$

We use boldface to denote N-dimensional vector fields, as well as their corresponding functional spaces, i.e. $\mathbf{H}^m = (H^m)^N$. We also define the divergence-free subspaces as follows:

$$\mathbf{V}^m = \{\mathbf{v} \in \mathbf{H}^m | \nabla\cdot\mathbf{v} = 0\}\,, \tag{1.7}$$

$$\mathbf{V}_0^m = \{\mathbf{v} \in \mathbf{H}_0^m | \nabla\cdot\mathbf{v} = 0\}\,, \quad m \geq 1. \tag{1.8}$$

We also consider the quotient space $L^2/\mathbb{R}$, which is equipped with the norm

$$\|f\|_{L^2/\mathbb{R}} = \inf_{c\in R} \|f - c\|\,. \tag{1.9}$$

A weak form of the problem reads: *find velocity* $\mathbf{u} \in \mathbf{H}^1$ *satisfying the initial and boundary conditions* (1.4),(1.3), *and pressure* $p \in L^2/\mathbb{R}$, *such that for all* $\mathbf{v} \in \mathbf{H}_0^1$ *and* $q \in L^2$, *the following hold:*

$$\left(\frac{\partial\mathbf{u}}{\partial t}, \mathbf{v}\right) + ((\mathbf{u}\cdot\nabla)\,\mathbf{u}, \mathbf{v}) + \frac{1}{Re}\,(\nabla\mathbf{u}, \nabla\mathbf{v}) - (p, \nabla\cdot\mathbf{v}) = (\mathbf{f}, \mathbf{v})\,, \tag{1.10}$$

$$(\nabla\cdot\mathbf{u}, q) = 0. \tag{1.11}$$

Clearly, if $(\mathbf{u}, p) \in \mathbf{H}^1 \times L^2/\mathbb{R}$ is a solution of the weak form, then the velocity is divergence-free. Moreover, $\mathbf{u}$ is the solution of the following solenoidal formulation: *find* $\mathbf{u} \in \mathbf{V}^1$ *satisfying the initial and boundary conditions* (1.4), (1.3), *such that for all* $\mathbf{v} \in \mathbf{V}_0^1$, *the following holds:*

$$\left(\frac{\partial\mathbf{u}}{\partial t}, \mathbf{v}\right) + ((\mathbf{u}\cdot\nabla)\,\mathbf{u}, \mathbf{v}) + \frac{1}{Re}\,(\nabla\mathbf{u}, \nabla\mathbf{v}) = (\mathbf{f}, \mathbf{v})\,. \tag{1.12}$$

1.3 Discretization

1.3.1 Spatial discretization

We will use two types of finite elements. $\mathbb{P}_1$ is the standard linear triangular/tetrahedral finite element with $(N+1)$ nodes at the vertices of the element. The functions associated with $\mathbb{P}_1$ are linear in each element, and, since their

values match at N points on each $(N-1)$-dimensional face, these functions are continuous everywhere in Ω. The linear triangular/tetrahedral finite element of Crouzeix-Raviart [15] type has its $(N+1)$ nodes placed at the midpoints/centroids of the faces. The functions are again linear, but match only at one point on each face. Thus the functions are not continuous across element boundaries. However, the continuity at the midpoint/centroid is enough to guarantee continuous flux, owing to the fact that a quadrature formula with one node gives the exact value of an integral of linear function over a simplex.

Let us now introduce a computational grid $\mathcal{G}_h^* = (\mathcal{N}_h, \mathcal{F}_h, \mathcal{T}_h)$ on Ω with grid parameter h. Here $\mathcal{N}_h$ is a collection of N_N nodes, $\mathcal{F}_h$ contains all N_F faces, and $\mathcal{T}_h$ is the set of N_T finite elements. The finite elements are N-dimensional simplices, i.e. they are triangles if N=2 and tetrahedra if N=3. The nodes in $\mathcal{N}_h$ are the vertices of the elements, and the faces in $\mathcal{F}_h$ are their $(N-1)$-dimensional sides. We also denote by $\mathcal{M}_h$ the set of midpoints/centroids of the faces.

Let us now define the following discrete spaces:

$$\mathbf{X}_h = \left\{ \mathbf{u}_h \in \mathbf{H}^1 |\ \mathbf{u}_h|_\tau \in \mathbb{P}_1^N, \forall \tau \in \mathcal{T}_h \right\}, \tag{1.13}$$

$$\mathbf{Y}_h = \left\{ \mathbf{u}_h \in \mathbf{L}^2 |\ \mathbf{u}_h|_\tau \in \mathbb{P}_1^N, \forall \tau \in \mathcal{T}_h, \right. \tag{1.14}$$
$$\left. \mathbf{u}_h \text{ is continuous at all } m \in \mathcal{M}_h \right\},$$

$$\mathbf{V}_h = \{ \mathbf{u}_h \in \mathbf{Y}_h |\ (\nabla \cdot \mathbf{u}_h)|_\tau = 0, \forall \tau \in \mathcal{T}_h \}, \tag{1.15}$$

$$Q_h = \left\{ q_h \in L^2 |\ q_h|_\tau \in \mathbb{P}_0, \forall \tau \in \mathcal{T}_h \right\}, \tag{1.16}$$

where $\mathbb{P}_n$ is the space of polynomials of degree n in N variables. For a given grid $\mathcal{G}_h$, $\mathbf{X}_h$ is the standard functional space associated with $\mathbb{P}_1$ elements, $\mathbf{Y}_h$ is the space of velocities of the linear Crouzeix-Raviart elements, $\mathbf{V}_h$ is the divergence-free subspace of $\mathbf{Y}_h$, and Q_h is the space of piecewise constant pressures.

Notice that the Crouzeix-Raviart element is nonconforming in the sense that $\mathbf{Y}_h$ is not a subspace of $\mathbf{H}^1$. This requires proper definitions of the weak discrete divergence and Laplace operators on $\mathbf{Y}_h$, which follow:

$$(\nabla \cdot \mathbf{u}_h, q_h) = \sum_{\tau \in \mathcal{T}_h} \int_\tau \nabla \cdot \mathbf{u}_h q_h \, dx \qquad \forall q_h \in Q_h, \tag{1.17}$$

$$(\nabla \mathbf{u}_h, \nabla \mathbf{v}_h) = \sum_{\tau \in \mathcal{T}_h} \int_\tau \nabla \mathbf{u}_h \cdot \nabla \mathbf{v}_h \, dx \qquad \forall \mathbf{v}_h \in \mathbf{Y}_h. \tag{1.18}$$

Obviously, these operators are extensions of the standard divergence and Laplace operators in $\mathbf{H}^1$ to $\mathbf{Y}_h$. Their consistency and stability are proved in [15]. The gradient operator in Q_h is the dual of the divergence operator

(1.17) in $\mathbf{Y}_h$:

$$(\nabla p_h, \mathbf{v}_h) = -\sum_{\tau \in \mathcal{T}_h} \int_\tau \nabla \cdot \mathbf{v}_h p_h \, dx \qquad \forall \mathbf{v}_h \in \mathbf{Y}_h. \tag{1.19}$$

1.3.2 Time splitting

Pressure-correction scheme. We now present a nonstandard pressure-correction scheme based on the two finite elements presented in section 1.3.1. We will restrict our attention to the first order scheme; extension to second order is straightforward. We start by dividing the time interval $[0, T]$ into subintervals of equal length Δt and denote the time levels by $t_n = n\Delta t, n = 0, 1, \ldots, \frac{T}{\Delta t}$. At each time level n we have two approximations for the velocity, $\tilde{\mathbf{u}}_h^n \in \mathbf{X}_h$ and $\mathbf{u}_h^n \in \mathbf{V}_h$, as well as an approximation for the pressure gradient, $\mathbf{g}_h^n \in \mathbf{Y}_h$. Assuming that $\mathbf{u}_h^0$ and $\mathbf{g}_h^0$ are proper approximations of the initial velocity and pressure gradient (see [36]), for $n = 0, 1, \ldots, \frac{T}{\Delta t} - 1$ we proceed as described below.

- Advection-diffusion step: *find* $\tilde{\mathbf{u}}_h^{n+1} \in \mathbf{X}_h$ *such that for all* $\tilde{\mathbf{v}}_h \in \mathbf{X}_h$,

$$\frac{1}{\Delta t}\left(\tilde{\mathbf{u}}_h^{n+1} - \mathbf{u}_h^n, \tilde{\mathbf{v}}_h\right) + \frac{1}{Re}\left(\nabla \tilde{\mathbf{u}}_h^{n+1}, \nabla \tilde{\mathbf{v}}_h\right) = \sum_{\tau \in \mathcal{T}_h} \int_\tau \left[\left(\mathbf{u}_h^n \cdot \nabla\right) \mathbf{u}_h^n\right] \cdot \tilde{\mathbf{v}}_h \, dx - \left(\mathbf{g}_h^n, \tilde{\mathbf{v}}_h\right) + \left(\mathbf{f}^{n+1}, \tilde{\mathbf{v}}_h\right). \tag{1.20}$$

  Here $\mathbf{f}^{n+1} = \mathbf{f}|_{t=t_{n+1}}$.

- Projection step: *find* $\mathbf{u}_h^{n+1} \in \mathbf{V}_h$ *such that for all* $\mathbf{v}_h \in \mathbf{V}_h$,

$$\left(\mathbf{u}_h^{n+1} - \tilde{\mathbf{u}}_h^{n+1}, \mathbf{v}_h\right) = 0. \tag{1.21}$$

  After $\mathbf{u}_h^{n+1}$ is computed, correct the pressure gradient $\mathbf{g}_h^{n+1} \in \mathbf{Y}_h$, so that

$$\mathbf{g}_h^{n+1} - \mathbf{g}_h^n = \frac{\tilde{\mathbf{u}}_h^{n+1} - \mathbf{u}_h^{n+1}}{\Delta t}. \tag{1.22}$$

  Note: the computation of (1.22) is not considered a separate step, since it does not require the solution of a linear system. Moreover, the pressure gradient is computed only weakly, tested in $\mathbf{X}_h$.

The quantity $\mathbf{g}_h^n$ is indeed an approximation for the pressure gradient. To see this, consider an alternative formulation of the projection step (1.21): *find*

$\mathbf{u}_h^{n+1} \in \mathbf{Y}_h$ *and* $\phi_h^{n+1} \in Q_h$ *such that for all* $\mathbf{v}_h \in \mathbf{Y}_h$ *and* $q_h \in Q_h$,,

$$\begin{aligned} &\left(\tilde{\mathbf{u}}_h^{n+1} - \mathbf{u}_h^{n+1}, \mathbf{v}_h\right) - \Delta t \sum_{\tau \in \mathcal{T}_h} \int_\tau \nabla \cdot \mathbf{v}_h \phi_h^{n+1} \, dx = 0, \\ &\sum_{\tau \in \mathcal{T}_h} \int_\tau \nabla \cdot \mathbf{u}_h^{n+1} q_h \, dx = 0. \end{aligned} \tag{1.23}$$

If we set $\phi_h^{n+1} = p_h^{n+1} - p_h^n$, we see that $\mathbf{g}_h^n$ is the Riesz representation in $\mathbf{Y}_h$ of the linear form

$$\mathbf{Y}_h \ni \mathbf{v}_h \mapsto \sum_{\tau \in \mathcal{T}_h} \int_\tau \nabla \cdot \mathbf{v}_h p_h^n \, dx \in \mathbb{R}.$$

This scheme can be extended to a rotational pressure-correction scheme by modifying the gradient correction equation (1.22) as follows:

$$\mathbf{g}_h^{n+1} - \mathbf{g}_h^n = \frac{\tilde{\mathbf{u}}_h^{n+1} - \mathbf{u}_h^{n+1}}{\Delta t} + \frac{1}{Re} \nabla \cdot \tilde{\mathbf{u}}_h^{n+1}. \tag{1.24}$$

The added divergence also acts as a stabilization term at higher Reynolds numbers and is less significant at lower ones.

This algorithm does not yield a proper pressure approximation. This is not surprising, since the general theory requires the spaces $\mathbf{X}_h$ and Q_h to be inf-sup stable, which is not the case here. The numerical results also confirm that the Lagrange multiplier ϕ_h^n in (1.23) (if computed) does not give a consistent approximation for the pressure correction. Therefore, the pressure must be computed separately in a postprocessing step. The solution we propose involves a Poisson problem, which adds to the computational cost of the algorithm; nevertheless, consistent pressure is not required by the algorithm itself and doesn't have to be computed at every timestep, but only if and when needed.

Consider the space of $\mathbb{P}_1$ continuous pressures

$$M_h = \left\{ q_h \in H^1 | \, q_h|_\tau \in \mathbb{P}_1, \forall \tau \in \mathcal{T}_h, \int_\Omega q_h \, dx = 0 \right\}.$$

We solve the problem: *find* $p_h^{n+1} \in M_h$ *such that for all* $q_h \in M_h$,

$$\begin{aligned} \left(\nabla p_h^{n+1}, \nabla q_h\right) = &-\frac{1}{\Delta t} \left(\tilde{\mathbf{u}}_h^{n+1} - \tilde{\mathbf{u}}_h^n, \nabla q_h\right) \\ &- \frac{1}{Re} \int_{\partial\Omega} \left(\mathbf{n} \times \nabla \times \tilde{\mathbf{u}}_h^{n+1}\right) \cdot \nabla q_h \, ds + \left(\mathbf{f}^{n+1}, \nabla q_h\right), \end{aligned} \tag{1.25}$$

where $\mathbf{n}$ is the unit outer normal vector to Ω. This equation is derived from the momentum equation (1.1) (less an advection term), tested with ∇q_h. The surface integral is a result of integration by parts of the diffusion term. We first apply the Helmholtz identity, taking into account the fact that the exact velocity is solenoidal, then integrate by parts, using the fact that any gradient is irrotational:

$$\begin{aligned}\int_\Omega \nabla^2\mathbf{u}\cdot\nabla q_h \, dx &= \int_\Omega [\nabla(\nabla\cdot\mathbf{u})]\cdot\nabla q_h \, dx - \int_\Omega [\nabla\times\nabla\times\mathbf{u}]\cdot\nabla q_h \, dx \\ &= -\int_\Omega (\nabla\times\mathbf{u})\cdot(\nabla\times\nabla q_h) \, dx - \int_{\partial\Omega} (\nabla\times\mathbf{u})\times(\nabla q_h)\cdot\mathbf{n} \, ds \\ &= -\int_{\partial\Omega} (\mathbf{n}\times\nabla\times\mathbf{u})\cdot\nabla q_h \, ds.\end{aligned}$$

Finally, $\tilde{\mathbf{u}}_n^{n+1}$ is used to approximate the exact velocity.

Velocity-correction scheme. The ideas presented above, applied to the velocity-correction schemes discussed in [39], yield the splitting procedure outlined below. Assuming $\tilde{\mathbf{u}}_h^0$ is a proper approximation of the initial condition $\mathbf{u}_0$, we proceed as follows.

- Projection step: *find* $\mathbf{u}_h^{n+1} \in \mathbf{Y}_h$ *and* $p_h^{n+1} \in Q_h$ *such that for all* $\mathbf{v}_h \in \mathbf{Y}_h$ *and* $q_h \in Q_h$,

$$\begin{aligned}\frac{1}{\Delta t}\left(\mathbf{u}_h^{n+1} - \tilde{\mathbf{u}}_h^n, \mathbf{v}_h\right) - \sum_{\tau\in\mathcal{T}_h}\int_\tau p_h^{n+1}\nabla\cdot\mathbf{v}_h \, dx &= \left(\mathbf{f}^{n+1}, \mathbf{v}_h\right) \\ - \left((\tilde{\mathbf{u}}_h^n\cdot\nabla)\,\tilde{\mathbf{u}}_h^n, \mathbf{v}_h\right) - \frac{1}{Re}\left(\nabla\tilde{\mathbf{u}}_h^n, \nabla\mathbf{v}_h\right), \quad (1.26) \\ \sum_{\tau\in\mathcal{T}_h}\int_\tau \nabla\cdot\mathbf{u}_h^{n+1} q_h \, dx = 0.\end{aligned}$$

- Correction step: *find* $\tilde{\mathbf{u}}_h^{n+1} \in \mathbf{X}_h$ *such that for all* $\tilde{\mathbf{v}}_h \in \mathbf{X}_h$,

$$\begin{aligned}\frac{1}{\Delta t}\left(\tilde{\mathbf{u}}_h^{n+1} - \tilde{\mathbf{u}}_h^n, \tilde{\mathbf{v}}_h\right) + \frac{1}{Re}\left(\nabla\tilde{\mathbf{u}}_h^{n+1}, \nabla\tilde{\mathbf{v}}_h\right) &= \left(\mathbf{f}^{n+1}, \tilde{\mathbf{v}}_h\right) \\ - \left((\tilde{\mathbf{u}}_h^n\cdot\nabla)\,\tilde{\mathbf{u}}_h^n, \tilde{\mathbf{v}}_h\right) + \left(p_h^{n+1}, \nabla\cdot\tilde{\mathbf{v}}_h\right). \quad (1.27)\end{aligned}$$

We modify slightly the correction step to remove the pressure from it. Notice that the first equation in (1.26) is also true in $\mathbf{X}_h$, since it is a subspace of $\mathbf{Y}_h$. This allows us to subtract from (1.27) the momentum equation in

(1.26) to arrive at the following equivalent correction step: *find* $\tilde{\mathbf{u}}_h^{n+1} \in \mathbf{X}_h$ *such that for all* $\tilde{\mathbf{v}}_h \in \mathbf{X}_h$,

$$\frac{1}{\Delta t}\left(\tilde{\mathbf{u}}_h^{n+1} - \mathbf{u}_h^{n+1}, \tilde{\mathbf{v}}_h\right) + \frac{1}{Re}\left(\nabla\left(\tilde{\mathbf{u}}_h^{n+1} - \tilde{\mathbf{u}}_h^{n}\right), \nabla\tilde{\mathbf{v}}_h\right) = 0. \tag{1.28}$$

Having done this, we can now rewrite the projection step (1.26) in solenoidal form: *find* $\mathbf{u}_h^{n+1} \in \mathbf{V}_h$ *such that for all* $\mathbf{v}_h \in \mathbf{V}_h$,

$$\frac{1}{\Delta t}\left(\mathbf{u}_h^{n+1} - \tilde{\mathbf{u}}_h^{n}, \mathbf{v}_h\right) = \left(\mathbf{f}^{n+1}, \mathbf{v}_h\right) - \left(\left(\tilde{\mathbf{u}}_h^{n}\cdot\nabla\right)\tilde{\mathbf{u}}_h^{n}, \mathbf{v}_h\right) - \frac{1}{Re}\left(\nabla\tilde{\mathbf{u}}_h^{n}, \nabla\mathbf{v}_h\right) \tag{1.29}$$

We treat the advection terms explicitly in order to keep the matrices symmetric and thus be able to apply the fast conjugate gradient method. Of course, this scheme is only conditionally stable. Unconditional stability can be achieved by using a semi-implicit advection (see [39]). In this case, the projection step remains the same, while the correction step becomes: *find* $\tilde{\mathbf{u}}_h^{n+1} \in \mathbf{X}_h$ *such that for all* $\tilde{\mathbf{v}}_h \in \mathbf{X}_h$,

$$\frac{1}{\Delta t}\left(\tilde{\mathbf{u}}_h^{n+1} - \mathbf{u}_h^{n+1}, \tilde{\mathbf{v}}_h\right) + \frac{1}{Re}\left(\nabla\left(\tilde{\mathbf{u}}_h^{n+1} - \tilde{\mathbf{u}}_h^{n}\right), \nabla\tilde{\mathbf{v}}_h\right) + \left(\left(\tilde{\mathbf{u}}_h^{n}\cdot\nabla\right)\left(\tilde{\mathbf{u}}_h^{n+1} - \tilde{\mathbf{u}}_h^{n}\right), \tilde{\mathbf{v}}_h\right) = 0 \tag{1.30}$$

1.4 Error analysis

The innovative idea in the proposed schemes concerns the solution of the generalized Stokes equation. For this reason, we will allow ourselves to omit the nonlinear advection term from the analysis. The analysis presented here is nearly direct application of the general framework developed in [36, 34, 33]. For simplicity we only consider the case of homogeneous Dirichlet boundary conditions. We also assume that the boundary of the domain Ω is smooth.

Preliminaries. For clarity, and following the notation used in [36], we define the divergence operators $B_h : \mathbf{X}_h \to Q_h$ and $C_h : \mathbf{Y}_h \to Q_h$ such that

$$\left(B_h\tilde{\mathbf{v}}_h, q_h\right) = \left(\nabla\cdot\tilde{\mathbf{v}}_h, q_h\right) \qquad \forall\tilde{\mathbf{v}}_h \in \mathbf{X}_h, q_h \in Q_h, \tag{1.31}$$

$$\left(C_h\mathbf{v}_h, q_h\right) = \sum_{\tau\in\mathcal{T}_h}\int_\tau \nabla\cdot\mathbf{v}_h q_h \, dx \qquad \forall\mathbf{v}_h \in \mathbf{Y}_h, q_h \in Q_h. \tag{1.32}$$

The proofs of the error estimates rely on the spaces $\mathbf{X}_h$ and Q_h satisfying a uniform inf-sup condition. It is clear that this is not the case here. We assume that a subspace $\hat{Q}_h \subseteq Q_h$ exists such that the pair $\left(\mathbf{X}_h, \hat{Q}_h\right)$ is inf-sup

stable. We also assume the existence of an orthogonal complement of $\hat{Q}_h$, i.e. a space $\tilde{Q}_h$ such that $(B_h\tilde{\mathbf{v}}_h, \tilde{q}_h) = 0$ for all $\tilde{q}_h \in \tilde{Q}_h$ and all $\tilde{\mathbf{v}}_h \in \mathbf{X}_h$. The existence of such subspaces has been established on the so called cross-grid (see, for example, [7]).

We now consider the following interpolation problem: *given* $\mathbf{u}(t)$ *and* $p(t)$, $t \in [0,T]$, *find* $\tilde{\mathbf{w}}_h(t) \in \mathbf{X}_h$ *and* $\hat{q}_h(t) \in \hat{Q}_h$ *such that for all* $\tilde{\mathbf{v}}_h \in \mathbf{X}_h$ *and all* $r_h \in Q_h$,

$$\begin{aligned} &(\nabla\tilde{\mathbf{w}}_h(t), \nabla\tilde{\mathbf{v}}_h) - (\hat{q}_h(t), \nabla\cdot\tilde{\mathbf{v}}_h) = (\nabla\mathbf{u}(t), \nabla\tilde{\mathbf{v}}_h) - (p(t), \nabla\cdot\tilde{\mathbf{v}}_h)\,, \\ &(\nabla\cdot\tilde{\mathbf{w}}_h(t), r_h) = (\nabla\cdot\mathbf{u}(t), r_h) \end{aligned} \tag{1.33}$$

If $\big(\mathbf{u}(t), p(t)\big)$ is the exact solution of (1.1) and (1.2), then $\mathbf{u}(t)$ is solenoidal, subsequently $\tilde{\mathbf{w}}_h(t)$ is pointwise divergence free. The existence of such piecewise linear interpolant for the velocity is vital for the convergence proof; however, it has only been proved for cross grids. Moreover, it is known that on diagonal grids the only divergence-free velocity in $\mathbf{X}_h$ is trivial: $\tilde{\mathbf{w}}_h(t) \equiv 0$, and the analysis presented here cannot be applied. In section 1.5.5 we present numerical evidence, showing that the scheme performs optimally on diagonal grids. This leads us to believe that the optimal error estimates can be extended to general grids, although the rigorous establishment of such claim remains elusive.

Lemma 1.1 (see [36, lemma 5.1]). *Let the solution* $\big(\mathbf{u}(t), p(t)\big)$ *of* (1.1), (1.2) *satisfy the regularity conditions* $\mathbf{u}(t) \in \mathbf{H}^2 \cap \mathbf{V}_0^1$ *and* $p(t) \in H^1 \cap L^2/\mathbb{R}$ *for* $t \in [0,T]$. *If* $\mathcal{G}_h$ *is a cross-grid, then there is a positive constant* c *independent of* h *such that*

$$\begin{aligned} \|\mathbf{u}(t) - \tilde{\mathbf{w}}_h(t)\|_0 + h\,\|\mathbf{u}(t) - \tilde{\mathbf{w}}_h(t)\|_1 + h\,\|p(t) - \hat{q}_h(t)\|_0 \\ \leq ch^2\big(\|\mathbf{u}(t)\|_2 + \|p(t)\|_1\big). \end{aligned} \tag{1.34}$$

Error estimates for the velocity. We need to verify that C_h^T is uniformly continuous with respect to the norm in H^1 (see [34, proposition 2.1]). For this purpose we define the projection $\Pi_h : L^1 \to Q_h$ such that for $q \in L^1$,

$$(\Pi_h q, r_h) = (q, r_h) \qquad \forall r_h \in Q_h.$$

Lemma 1.2. *There is a positive constant* c *independent of* h *such that*

$$\big\|C_h^T \Pi_h q\big\|_0 \leq c\,\|q\|_1 \qquad \forall q \in H^1.$$

Proof. According to the definition of C_h, the following holds true:

$$\begin{aligned}\left\|C_h^T \Pi_h q\right\|_0^2 &= \left(C_h^T \Pi_h q, C_h^T \Pi_h q\right)_0 = -\sum_{\tau\in\mathcal{T}_h}\int_\tau \Pi_h q \nabla\cdot C_h^T \Pi_h q \, dx \\ &= -\sum_{\tau\in\mathcal{T}_h}\int_{\partial\tau} \Pi_h q \left(C_h^T \Pi_h q\right)\cdot\mathbf{n}\, ds.\end{aligned}$$

For each internal face $f_i \in \mathcal{F}_{hi}$, we denote by $\bar{q}_i$ the mean value of the restriction of q to f_i. Let $\bar{q}$ be the function on $\mathcal{F}_{hi}$, whose value on f_i equals $\bar{q}_i$. On each interior face f_i, we have (see [15, Hypothesis H.2 and Example 4])

$$\int_{f_i} \bar{q}_i \left[\mathbf{v}\cdot\mathbf{n}_i\right] \, ds = 0 \qquad \forall \mathbf{v} \in \mathbf{Y}_h,$$

and on each boundary face we have

$$\int_{f_i} \bar{q}_i \mathbf{v}\cdot\mathbf{n}_i \, ds = 0 \qquad \forall \mathbf{v} \in \mathbf{Y}_h,$$

due to Dirichlet boundary conditions. The symbol $\left[\mathbf{v}\cdot\mathbf{n}_i\right]$ denotes the jump of the normal component of $\mathbf{v}$. Therefore,

$$\begin{aligned}\left\|C_h^T \Pi_h q\right\|_0^2 &= -\sum_{\tau\in\mathcal{T}_h}\int_{\partial\tau} \left(\Pi_h q - \bar{q}\right)\left(C_h^T \Pi_h q\right)\cdot\mathbf{n}\, ds \\ &\le \sum_{\tau\in\mathcal{T}_h} \left\|\Pi_h q - \bar{q}\right\|_{L^2(\partial\tau)} \left\|C_h^T \Pi_h q\right\|_{L^2(\partial\tau)}.\end{aligned}$$

Now, using the mesh regularity together with standard scaling arguments and Deny-Lions lemma, we infer that

$$\begin{aligned}\left\|C_h^T \Pi_h q\right\|_0^2 &\le \sum_{\tau\in\mathcal{T}_h} c h^{1/2} \left\|q\right\|_{H^1(\tau)} h^{-1/2} \left\|C_h^T \Pi_h q\right\|_{L^2(\tau)} \\ &\le c \left\|q\right\|_1 \left\|C_h^T \Pi_h q\right\|_0.\end{aligned}$$

The proof is complete. □

Let E be a normed space with norm $\|\cdot\|_E$, and let $a^n \in E$ for $n = 0, \ldots, \frac{T}{\Delta t}$. We denote

$$\left\|a^n\right\|_{\ell^2(E)} = \left(\Delta t \sum_{n=0}^{T/\Delta t} \left\|a^n\right\|_E^2\right)^{1/2}, \qquad \left\|a^n\right\|_{\ell^\infty(E)} = \max_{0\le n\le T/\Delta t} \left\|a^n\right\|_E.$$

Theorem 1.3. *Let the solution* $\big(\mathbf{u}(t), p(t)\big)$ *of* (1.1), (1.2) *satisfy the regularity conditions* $\mathbf{u}(t) \in \mathbf{H}^2 \cap \mathbf{V}_0^1$ *and* $p(t) \in H^1 \cap L^2/\mathbb{R}$ *for* $t \in [0,T]$. *There is a positive constant* c *independent of* h *such that the velocity approximations* $\tilde{\mathbf{u}}_h^n$ *and* $\mathbf{u}_h^n$ *produced by scheme* (1.20), (1.21) *satisfy*

$$\|\mathbf{u}^n - \tilde{\mathbf{u}}_h^n\|_{\ell^\infty(\mathbf{L}^2)} + \|\mathbf{u}^n - \mathbf{u}_h^n\|_{\ell^\infty(\mathbf{L}^2)} \le c\big(h^2 + \Delta t\big),$$
$$\|\mathbf{u}^n - \mathbf{u}_h^n\|_{\ell^2(\mathbf{H}^1)} \le c\big(h + \Delta t\big).$$

Proof. The proof follows [33, 36, 34] closely. □

Convergence of the continuous approximation of the pressure. In this paragraph we prove the optimal convergence of the pressure produced at the postprocessing step (1.25). Denote $\tilde{\mathbf{e}}_h^n = \mathbf{u}^n - \tilde{\mathbf{u}}_h^n$ and $\mathbf{e}_h^n = \mathbf{u}^n - \mathbf{u}_h^n$. We will use the following result.

Lemma 1.4 (see [36, lemma 5.6]). *If the algorithm* (1.20), (1.21) *is properly initialized, then*

$$\big\|\tilde{\mathbf{e}}_h^{n+1} - \tilde{\mathbf{e}}_h^n\big\|_{\ell^2(\mathbf{H}^1)} \le c\Delta t\big(\Delta t + h\big).$$

The accuracy of the $\mathbb{P}_1$ approximation for the pressure is subject to the following error estimate.

Theorem 1.5. *Under the assumptions of theorem 1.3 and lemma 1.4, there is a positive constant* c *independent of* h *such that the solution* $p_h^n \in M_h$ *of* (1.25) *satisfies*

$$\|p^n - p_h^n\|_{\ell^2(L^2/\mathbb{R})} \le c\big(\Delta t + h\big).$$

Proof. We take the exact momentum equation (1.1) at time level t_{n+1} and test it with ∇q_h, for $q_h \in M_h$. After ignoring the advection term and integrating by parts the diffusion term (the same way as we did in the derivation of (1.25)), we arrive at

$$\big(\nabla p^{n+1}, \nabla q_h\big) = -\left(\frac{\partial \mathbf{u}^{n+1}}{\partial t}, \nabla q_h\right) - \frac{1}{Re}\int_{\partial\Omega} \big(\mathbf{n}\times\nabla\times\mathbf{u}^{n+1}\big)\cdot\nabla q_h \, ds + \big(\mathbf{f}^{n+1}, \nabla q_h\big). \quad (1.35)$$

Subtracting (1.25) from (1.35) produces

$$\big(\nabla(p^{n+1} - p_h^{n+1}), \nabla q_h\big) = -\frac{1}{\Delta t}\big(\tilde{\mathbf{e}}_h^{n+1} - \tilde{\mathbf{e}}_h^n, \nabla q_h\big) - \frac{1}{Re}\int_{\partial\Omega} \big(\mathbf{n}\times\nabla\times\tilde{\mathbf{e}}_h^{n+1}\big)\cdot\nabla q_h \, ds - \left(\frac{\partial \mathbf{u}^{n+1}}{\partial t} - \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t}, \nabla q_h\right). \quad (1.36)$$

Let $\hat{p}_h^{n+1}$ be the piecewise linear Lagrange interpolant of p^{n+1}, which clearly satisfies

$$\left\|\nabla(p^{n+1}-\hat{p}_h^{n+1})\right\|_0 \le ch. \tag{1.37}$$

We denote the error in the pressure by $\varepsilon_h^{n+1}=\hat{p}_h^{n+1}-p_h^{n+1}$. Incorporating ε_h^{n+1} into (1.36), we have

$$\begin{aligned}\left(\nabla\varepsilon_h^{n+1},\nabla q_h\right) = &-\frac{1}{\Delta t}\left(\tilde{\mathbf{e}}_h^{n+1}-\tilde{\mathbf{e}}_h^{n},\nabla q_h\right)-\frac{1}{Re}\int_{\partial\Omega}\left(\mathbf{n}\times\nabla\times\tilde{\mathbf{e}}_h^{n+1}\right)\cdot\nabla q_h\,ds\\ &-\left(\frac{\partial\mathbf{u}^{n+1}}{\partial t}-\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t},\nabla q_h\right)-\left(\nabla(p^{n+1}-\hat{p}_h^{n+1}),\nabla q_h\right).\end{aligned} \tag{1.38}$$

Obviously $\varepsilon_h^{n+1}\in M_h$, so we are allowed to set $q_h=\varepsilon_h^{n+1}$ in (1.38) to obtain

$$\begin{aligned}\left\|\nabla\varepsilon_h^{n+1}\right\|^2 = &-\frac{1}{\Delta t}\left(\tilde{\mathbf{e}}_h^{n+1}-\tilde{\mathbf{e}}_h^{n},\nabla\varepsilon_h^{n+1}\right)-\frac{1}{Re}\int_{\partial\Omega}\left(\mathbf{n}\times\nabla\times\tilde{\mathbf{e}}_h^{n+1}\right)\cdot\nabla\varepsilon_h^{n+1}\,ds\\ &-\left(\frac{\partial\mathbf{u}^{n+1}}{\partial t}-\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t},\nabla\varepsilon_h^{n+1}\right)-\left(\nabla(p^{n+1}-\hat{p}_h^{n+1}),\nabla\varepsilon_h^{n+1}\right).\end{aligned} \tag{1.39}$$

Now we apply the Cauchy-Schwarz inequality on every term in the right-hand side:

$$\begin{aligned}\left\|\nabla\varepsilon_h^{n+1}\right\|^2 \le &\frac{1}{\Delta t}\left\|\tilde{\mathbf{e}}_h^{n+1}-\tilde{\mathbf{e}}_h^{n}\right\|\left\|\nabla\varepsilon_h^{n+1}\right\|+\frac{1}{Re}\left\|\nabla\times\tilde{\mathbf{e}}_h^{n+1}\right\|_{\mathbf{L}^2(\partial\Omega)}\left\|\nabla\varepsilon_h^{n+1}\right\|_{\mathbf{L}^2(\partial\Omega)}\\ &+\left\|\frac{\partial\mathbf{u}^{n+1}}{\partial t}-\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t}\right\|\left\|\nabla\varepsilon_h^{n+1}\right\|+\left\|\nabla(p^{n+1^*}-\hat{p}_h^{n+1})\right\|\left\|\nabla\varepsilon_h^{n+1}\right\|.\end{aligned} \tag{1.40}$$

The last equation, together with (1.37), lemma 1.4 and inverse inequality yield

$$\left\|\nabla\varepsilon_h^{n+1}\right\|_0 \le c\left(\Delta t+h+h^{-1}\left\|\tilde{\mathbf{e}}_h^{n+1}\right\|_1\right). \tag{1.41}$$

This result, together with theorem 1.3 and a standard duality argument give optimal estimate in L^2 norm:

$$\left\|\varepsilon_h^{n+1}\right\|_{\ell^2(L^2)} \le c\left(\Delta t+h\right). \qquad \square$$

1.5 Implementation in two spatial dimensions

In this section N=2.

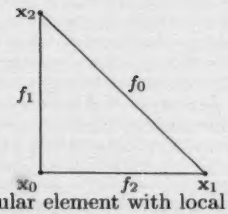

Figure 1: Reference triangular element with local indices of nodes and faces.

1.5.1 General remarks

It was already mentioned in 1.3.1 that we maintain a grid $\mathcal{G}_h$ consisting of nodes, faces and elements.

The nodes consist of two coordinates and a boundary condition code, and each node is assigned a global index starting from 0. Nodes in the interior of Ω are assigned indices from 0 to $N_{N_i} - 1$, and boundary nodes (the ones where essential boundary condition is prescribed) get the highest indices, from N_{N_i} to $N_N - 1$. This is done to facilitate the imposition of boundary conditions.

Whenever we need to invert a matrix A, where essential boundary conditions are prescribed, we replace the equations corresponding to boundary nodes with equations that have one on the main diagonal and the prescribed value in the right-hand side. These equations don't actually need to be solved, and we don't store them. We store the rest of the matrix in two parts – the main part is square and corresponds to the interior nodes and the other part corresponds to rows of interior nodes and columns of boundary nodes. We multiply the prescribed boundary values by the second matrix, subtract the result from the right-hand side, and invert the first sub-matrix as demonstrated below:

$$\begin{pmatrix} A_{ii} & A_{ib} \\ 0 & I \end{pmatrix} \begin{pmatrix} X_i \\ X_b \end{pmatrix} = \begin{pmatrix} F_i \\ X_{\text{b.cond.}} \end{pmatrix} \Longrightarrow \begin{cases} X_i = A_{ii}^{-1} \left(F_i - A_{ib} X_{\text{b.cond.}} \right) \\ X_b = X_{\text{b.cond.}} \end{cases}$$

Each element in the grid contains three nodes and three faces. The nodes have local indices 0, 1 and 2, going in counter-clockwise direction. The local indices of faces are assigned so that face i is always opposite node i (Figure 1). We do not allow an element to have all of its nodes on the boundary.

The faces in the grid contain two nodes and two elements. The node with smaller global index is always taken first (local index 0). The faces are sorted by the indices of their nodes. Because the nodes are sorted so that boundary nodes have larger indices, the faces on the boundary of Ω will have largest indices as well. The two element indices stored with each face are the elements to which this face belongs and are taken in no particular order.

Faces on the boundary belong to only one element. We also maintain two unit vectors associated with each face – tangent and normal. The tangent is always the vector pointing from node 0 towards node 1, normalized to unit length. The normal vector is computed by rotating the tangent vector by $\frac{\pi}{2}$ clockwise. With this setup, it is easy to obtain the outer normal vector to the side i of an element: we take the normal vector to the face with local index i and flip its sign if, and only if, the global index of node $\big[(i+1) \bmod 3\big]$ is greater than the global index of node $\big[(i+2) \bmod 3\big]$.

The local mass matrices of the two-dimensional Crouzeix-Raviart elements are diagonal. This is due to the fact that the interpolation nodes (the midpoints of the faces) are also the nodes of a Gaussian quadrature formula, which is exact for integration of second degree polynomials in a triangle. As a result, the global mass matrix is also diagonal – this fact will play a significant role in the implementation of the projection.

The implementation of any computation involving the standard $\mathbb{P}_1$ element is straightforward. Moreover, often in a projection method the bulk of the computational time is used in the projection step. Thus, as we try to speed up the solution, we will allocate most of our efforts to finding the fastest projection algorithm. Frequently, the projection step is implemented as a Poisson problem for the pressure. In our setting, this is not straightforward, as the pressure is discontinuous. In the following sections we discuss and compare several possibilities. We will consider (1.21) and (1.23), although extension to (1.26) and (1.29) is direct.

In the sequel, we will denote by $\{\varphi_i\}_{i=0}^{N_N-1}$ the usual Lagrangian interpolation basis of the $\mathbb{P}_1$ element. There is one basis function associated with each node in the grid:

$$\begin{aligned} \varphi_i|_\tau &\in \mathbb{P}_1 && \forall \tau \in \mathcal{T}_h, \\ \varphi_i(\mathbf{x}_j) &= \delta_{ij} && \forall \mathbf{x}_j \in \mathcal{N}_h. \end{aligned}$$

Any $\tilde{\mathbf{u}}_h \in \mathbf{X}_h$ can be written as

$$\tilde{\mathbf{u}}_h = \sum_{i=0}^{N_N-1} \left[\tilde{U}_{x,i}\mathbf{e}_x + \tilde{U}_{y,i}\mathbf{e}_y\right] \varphi_i, \tag{1.42}$$

where $\mathbf{e}_x$ and $\mathbf{e}_y$ are the unit vectors in direction x and y respectively, and $\tilde{U}_{x,i}$ and $\tilde{U}_{y,i}$ are the values of the x and y components of $\tilde{\mathbf{u}}_h$ at node $\mathbf{x}_i \in \mathcal{N}_h$.

Similarly, we will use $\{\psi_i\}_{i=0}^{N_F-1}$ to denote the Lagrangian interpolation basis of the Crouzeix-Raviart element. This time the functions correspond to

faces:

$$\begin{aligned} \psi_i|_\tau &\in \mathbb{P}_1 && \forall \tau \in \mathcal{T}_h, \\ \psi_i(m_j) &= \delta_{ij} && \forall m_j \in \mathcal{M}_h. \end{aligned}$$

The expansion of $\mathbf{u}_h \in \mathbf{Y}_h$ in this Crouzeix-Raviart basis reads

$$\mathbf{u}_h = \sum_{i=0}^{N_F-1} \left[U_{x,i}^{CR} \mathbf{e}_x + U_{y,i}^{CR} \mathbf{e}_y \right] \psi_i,$$

where $U_{x,i}^{CR}$ and $U_{y,i}^{CR}$ are the x and y components of $\mathbf{u}_h$ at the midpoint m_i of face $f_i \in \mathcal{F}_h$. Since $\mathbf{X}_h \subset \mathbf{Y}_h$, we can also write the expansion of $\tilde{\mathbf{u}}_h$ in the $\{\psi_i\}$ basis:

$$\tilde{\mathbf{u}}_h = \sum_{i=0}^{N_F-1} \left[\tilde{U}_{x,i}^{CR} \mathbf{e}_x + \tilde{U}_{y,i}^{CR} \mathbf{e}_y \right] \psi_i. \tag{1.43}$$

$\tilde{U}^{CR}$ can be evaluated easily as the average of the values of $\tilde{U}$ at the two endpoints of the corresponding face. Alternatively, the expansions in Crouzeix-Raviart basis can be written in terms of tangential and normal components as

$$\mathbf{u}_h = \sum_{i=0}^{N_F-1} \left[U_{t,i}^{CR} \mathbf{t}_i + U_{n,i}^{CR} \mathbf{n}_i \right] \psi_i, \tag{1.44}$$

$$\tilde{\mathbf{u}}_h = \sum_{i=0}^{N_F-1} \left[\tilde{U}_{t,i}^{CR} \mathbf{t}_i + \tilde{U}_{n,i}^{CR} \mathbf{n}_i \right] \psi_i. \tag{1.45}$$

The linear system resulting from the projection substep of any of the projection algorithms described above can be resolved in several different ways. We considered the following three possibilities.

1.5.2 Lagrange multipliers

With the pressure being piecewise constant and the velocity being piecewise linear, the incompressibility constraint actually imposes the requirement that the divergence of the velocity must be identically 0 in each element. We formulate the following constrained minimization problem: *find* $\mathbf{u}_h \in \mathbf{Y}_h$ *such that,*

$$\left| \begin{aligned} &\|\mathbf{u}_h - \tilde{\mathbf{u}}_h\| \longrightarrow \min \\ &\int_\tau \nabla \cdot \mathbf{u}_h \, dx = 0 \qquad \forall \tau \in \mathcal{T}_h. \end{aligned} \right. \tag{1.46}$$

We introduce one Lagrange multiplier λ_τ for each element $\tau \in \mathcal{T}_h$ and solve the following: *find* $\mathbf{u}_h \in \mathbf{Y}_h$ *and constants* $\lambda_\tau, \tau \in \mathcal{T}_h$ *such that:*

$$\|\mathbf{u}_h - \tilde{\mathbf{u}}_h\| + \sum_{\tau \in \mathcal{T}_h} \lambda_\tau \int_\tau \nabla \cdot \mathbf{u}_h \, dx \longrightarrow \min . \tag{1.47}$$

This problem is equivalent to (1.23), if we define

$$\phi_h = \frac{1}{\Delta t} \sum_{\tau \in \mathcal{T}_h} 1_\tau \lambda_\tau, \tag{1.48}$$

where 1_τ is the indicator function of the element τ. The problem (1.47) leads to the following linear system, written in terms of normal and tangential components:

$$\begin{pmatrix} M & 0 & 0 \\ 0 & M & D^T \\ 0 & D & 0 \end{pmatrix} \begin{pmatrix} U_t^{CR} \\ U_n^{CR} \\ \Lambda \end{pmatrix} = \begin{pmatrix} M\tilde{U}_t^{CR} \\ M\tilde{U}_n^{CR} \\ 0 \end{pmatrix}. \tag{1.49}$$

In (1.49) M is the diagonal mass matrix of Crouzeix-Raviart and D is the matrix of the discrete divergence operator. It is clear that the tangential components are decoupled from the system:

$$\begin{aligned} & U_t^{CR} = \tilde{U}_t^{CR}, \\ & \begin{pmatrix} M & D^T \\ D & 0 \end{pmatrix} \begin{pmatrix} U_n^{CR} \\ \Lambda \end{pmatrix} = \begin{pmatrix} M\tilde{U}_n^{CR} \\ 0 \end{pmatrix}. \end{aligned} \tag{1.50}$$

We impose the boundary conditions on the normal components by replacing the corresponding equations:

$$\begin{pmatrix} M_i & 0 & D_i^T \\ 0 & I & 0 \\ D_i & D_b & 0 \end{pmatrix} \begin{pmatrix} U_{ni}^{CR} \\ U_{nb}^{CR} \\ \Lambda \end{pmatrix} = \begin{pmatrix} M\tilde{U}_{ni}^{CR} \\ \tilde{U}_{nb}^{CR} \\ 0 \end{pmatrix}. \tag{1.51}$$

To avoid solving a saddle point problem by the slow Uzawa iteration, we construct the Schur complement of M_i by multiplying the first row by $D_i M_i^{-1}$, the second row by D_b, and subtracting both from the last row.

$$\begin{aligned} & U_t^{CR} = \tilde{U}_t^{CR}, \\ & U_{nb}^{CR} = \tilde{U}_{nb,}^{CR} \\ & \begin{pmatrix} M_i & D_i^T \\ 0 & -D_i M_i^{-1} D_i^T \end{pmatrix} \begin{pmatrix} U_{ni}^{CR} \\ \Lambda \end{pmatrix} = \begin{pmatrix} M_i \tilde{U}_{ni}^{CR} \\ -D\tilde{U}_n^{CR} \end{pmatrix}. \end{aligned} \tag{1.52}$$

Now it is clear that the solution of the problem is given by

$$\begin{aligned} U_t^{CR} &= \tilde{U}_t^{CR}, \\ U_{nb}^{CR} &= \tilde{U}_{nb}^{CR}, \\ \Lambda &= \left(D_i M_i^{-1} D_i^T\right)^{-1} D\tilde{U}_n^{CR}, \\ U_{ni}^{CR} &= \tilde{U}_{ni}^{CR} - M_i^{-1} D_i^T \Lambda. \end{aligned} \tag{1.53}$$

Unlike other projection methods with discontinuous pressure, we can construct the Schur complement $S = D_i M_i^{-1} D_i^T$ explicitly, due to the fact that M is diagonal. This saves us the time normally needed to invert M_i on every iteration of the conjugate gradient method.

The matrix S is singular, because we imposed a boundary condition on U^{CR} that prescribes the total divergence of $\mathbf{u}_h$ according to

$$\int_\Omega \nabla \cdot \mathbf{u}_h \, dx = \int_{\partial\Omega} \mathbf{u}_h \cdot \mathbf{n} \, ds. \tag{1.54}$$

This makes the side constraints imposed by the λ-s linearly dependent. If we set the divergence to 0 in all elements but one, the divergence in this last element is assigned by the boundary condition. The remedy is simply to remove the last row and the last column from the matrix. The new matrix has dimensions $(N_T - 1) \times (N_T - 1)$ and is symmetric and positive definite. In order to obtain a divergence-free solution, we must always ensure that the boundary condition satisfies

$$\int_{\partial\Omega} \mathbf{u}_{hb} \cdot \mathbf{n} \, ds = 0. \tag{1.55}$$

1.5.3 Inter-elemental multipliers

In this section we consider the linear system in (1.50) and solve it by relaxing the natural continuity on the velocity flux through element boundaries and imposing it via an additional set of side constraints. This approach to the solution of Poisson problems is widely used with mixed methods for elliptic problems, as described in [7, p.178].

Let us define the following functional spaces:

$$\mathbf{Z}_h = \left\{ \mathbf{u}_h \in \mathbf{L}^2 |\ \mathbf{u}_h|_\tau \in \mathbb{P}_1^N, \forall \tau \in \mathcal{T}_h \right\}, \tag{1.56}$$

$$\mathbf{W}_h = \left\{ \mathbf{u}_h \in \mathbf{Z}_h |\ (\nabla \cdot \mathbf{u}_h)|_\tau = 0, \forall \tau \in \mathcal{T}_h \right\}. \tag{1.57}$$

The vector fields in $\mathbf{Z}_h$ are piecewise linear in each element and in general completely discontinuous across element boundaries. $\mathbf{W}_h$ is the divergence-

free subspace of $\mathbf{Z}_h$. Let $\bar{\mathbf{u}}_h \in \mathbf{Z}_h$. Since functions in $\mathbf{Z}_h$ are piecewise linear, they have three degrees of freedom per element, i.e. $\dim \mathbf{Z}_h = 3N_T$. For the sake of convenience, we place the degrees of freedom in the midpoints of the sides, thus on each interior $m \in \mathcal{M}_h$, each component of $\bar{\mathbf{u}}_h$ will have two values – one for each element containing m. We will use $\left[\bar{\mathbf{u}}_h(m_i)\right]$ to indicate the jumps of $\bar{\mathbf{u}}_h$ at points $m_i \in \mathcal{M}_h, m_i \notin \partial\Omega$.

Let us now introduce the following notation for the Lagrangian interpolation basis of $\mathbf{Z}_h$: for each element $\tau \in \mathcal{T}_h$ and for each element face $i \in \{0,1,2\}$, we have one basis function $\psi_{\tau,i}$ such that

$$\begin{aligned} \psi_{\tau,i}|_{\tilde{\tau}} &= 0 \qquad \forall \tilde{\tau} \neq \tau, \tilde{\tau} \in \mathcal{T}_h, \\ \psi_{\tau,i}|_{\tau} \in \mathbb{P}_1 \qquad &\text{with} \qquad \psi_{\tau,i}(m_{\tau,j}) = \delta_{ij}, \end{aligned}$$

where $m_{\tau,j}, j \in \{0,1,2\}$ are the midpoints of the sides of τ. Here and for the remainder of this section, subscript 'τ,i' refers to side $i \in \{0,1,2\}$ of element $\tau \in \mathcal{T}_h$. Since the inclusions $\mathbf{X}_h \subset \mathbf{Y}_h \subset \mathbf{Z}_h$ hold, we can write the expansions of $\tilde{\mathbf{u}}_h$ and $\mathbf{u}_h$ in the tangent-normal basis of $\mathbf{Z}_h$:

$$\mathbf{u}_h = \sum_{\tau \in \mathcal{T}_h} \sum_{i=0}^{2} \Big(U_{t,\tau,i} \mathbf{t}_{\tau,i} + U_{n,\tau,i} \mathbf{n}_{\tau,i} \Big) \psi_{\tau,i}, \tag{1.58}$$

$$\tilde{\mathbf{u}}_h = \sum_{\tau \in \mathcal{T}_h} \sum_{i=0}^{2} \Big(\tilde{U}_{t,\tau,i} \mathbf{t}_{\tau,i} + \tilde{U}_{n,\tau,i} \mathbf{n}_{\tau,i} \Big) \psi_{\tau,i}. \tag{1.59}$$

The projection step can be written as the following minimization problem in $\mathbf{Z}_h$: *find* $\mathbf{u}_h \in \mathbf{Z}_h$ *such that*

$$\left| \begin{aligned} &\|\mathbf{u}_h - \tilde{\mathbf{u}}_h\| \longrightarrow \min, \\ &\int_\tau \nabla \cdot \mathbf{u}_h \, dx = 0 \qquad \forall \tau \in \mathcal{T}_h, \\ &\left[\mathbf{u}_h(m_i)\right] = 0 \qquad \forall m_i \in \mathcal{M}_h, m_i \notin \partial\Omega. \end{aligned} \right. \tag{1.60}$$

After introducing Lagrange multipliers λ_τ as in section 1.5.2, and two more sets of Lagrange multipliers α_{t,m_i} and α_{n,m_i} to enforce continuity of the tangential and normal components respectively, we arrive at the following equivalent unconstrained formulation: *find* $\mathbf{u}_h \in \mathbf{Z}_h$ *and constants* $\lambda_\tau, \alpha_{t,m_i}$ *and* $\alpha_{n,m_i}, \tau \in \mathcal{T}_h, m_i \in \mathcal{M}_h, m_i \notin \partial\Omega$ *such that*

$$\|\mathbf{u}_h - \tilde{\mathbf{u}}_h\| + \sum_{\tau \in \mathcal{T}_h} \lambda_\tau \int_\tau \nabla \cdot \mathbf{u}_h \, dx$$

$$+ \sum_{\substack{m_i \in \mathcal{M}_h \\ m_i \notin \partial\Omega}} \left(\alpha_{t,m_i} [\mathbf{u}_h(m_i)] \cdot \mathbf{t}_i + \alpha_{n,m_i} [\mathbf{u}_h(m_i)] \cdot \mathbf{n}_i \right) \longrightarrow \min . \quad (1.61)$$

Similarly to (1.50), we can see that the tangential components are decoupled from the other unknowns and are unchanged by the projection. The remaining linear system for the normal components is

$$\begin{pmatrix} \bar{M} & \bar{D}^T & C^T \\ \bar{D} & 0 & 0 \\ C & 0 & 0 \end{pmatrix} \begin{pmatrix} U_n \\ \Lambda \\ \Lambda_n \end{pmatrix} = \begin{pmatrix} \bar{M}\tilde{U}_n \\ 0 \\ 0 \end{pmatrix}, \quad (1.62)$$

where $\bar{M}$ and $\bar{D}$ are respectively the mass matrix and the divergence operator in $\mathbf{Z}_h$, and C is a matrix of the "jump" operator. At first glance, this system seems much worse to solve than (1.50), but we shall see that it can be greatly simplified.

Let us first look at the problem

$$\begin{pmatrix} \bar{M} & \bar{D}^T \\ \bar{D} & 0 \end{pmatrix} \begin{pmatrix} U_n \\ \Lambda \end{pmatrix} = \begin{pmatrix} \bar{M}\tilde{U}_n \\ 0 \end{pmatrix}. \quad (1.63)$$

Each row of $\bar{D}$ corresponds to one element τ and has nonzero entries only in the three columns corresponding to the faces of τ. The mass matrix is block diagonal, with 3×3 blocks on the diagonal being the local Crouzeix-Raviart mass matrices of the elements in $\mathcal{T}_h$. Thus (1.63) actually consists of N_T independent problems of local projections – one for each element. The local projection is a 4×4 system in the form

$$\begin{pmatrix} M_\tau & D_\tau^T \\ D_\tau & 0 \end{pmatrix} \begin{pmatrix} U_{n,\tau} \\ \lambda_\tau \end{pmatrix} = \begin{pmatrix} M_\tau \tilde{U}_{n,\tau} \\ 0 \end{pmatrix}, \quad (1.64)$$

which can be inverted easily. We can write the solution for $U_{n,\tau}$ symbolically as

$$U_{n,\tau} = \mathcal{P}_\tau \tilde{U}_{n,\tau}, \quad (1.65)$$

where $\mathcal{P}_\tau$ is the matrix of the projection in $\operatorname{span}\{\mathbf{n}_{\tau,i}\psi_{\tau,i} | i = 0,1,2\}$ onto its divergence-free subspace. If τ has a face on $\partial\Omega$, we impose the boundary condition in the local system (1.64), and the resulting local projection matrix accounts for it. These 3×3 local projection matrices can be assembled into one block diagonal matrix $\mathcal{P}$ of the projection in $\mathbf{Z}_h$ onto $\mathbf{W}_h$. Thus

$$U_n = \mathcal{P}\tilde{U}_n \quad (1.66)$$

is the velocity solution of (1.63).

Now we go back to (1.62) and apply the same approach to eliminate Λ. The system reduces to

$$\begin{pmatrix} I & 0 & \mathcal{P}C^T \\ 0 & \bar{D}\bar{M}^{-1}\bar{D}^T & \bar{D}\bar{M}^{-1}C^T \\ C & 0 & 0 \end{pmatrix} \begin{pmatrix} U_n \\ \Lambda \\ A_n \end{pmatrix} = \begin{pmatrix} \mathcal{P}\tilde{U}_n \\ \bar{D}\tilde{U}_n \\ 0 \end{pmatrix}. \tag{1.67}$$

Obviously, the equations for Λ are now decoupled:

$$\begin{pmatrix} I & \mathcal{P}C^T \\ C & 0 \end{pmatrix} \begin{pmatrix} U_n \\ A_n \end{pmatrix} = \begin{pmatrix} \mathcal{P}\tilde{U}_n \\ 0 \end{pmatrix}. \tag{1.68}$$

Using the Schur complement of I, we express the solution as

$$\begin{aligned} A_n &= \left(C\mathcal{P}C^T\right)^{-1} C\mathcal{P}\tilde{U}_n, \\ U_n &= \mathcal{P}\left(\tilde{U}_n - C^T A_n\right). \end{aligned} \tag{1.69}$$

The Schur complement $\bar{S} = C\mathcal{P}C^T$ can be constructed explicitly, since $\mathcal{P}$ is block diagonal and C has a very simple structure – it has two nonzero entries in each row: 1 and -1. $\bar{S}$ is symmetric, even though the matrix $\mathcal{P}$ is not. The local projection matrix is not symmetric for elements having a face on the boundary. Regardless, $\bar{S}$ is symmetric, since the columns of C corresponding to boundary faces contain all zeros, effectively eliminating asymmetric rows and columns of $\mathcal{P}$ from the Schur complement.

The matrix $\bar{S}$ has dimensions equal to the number of interior faces, which is significantly greater than the dimension of the system in the previous section. In the two-dimensional case we have $N_F \approx \frac{3}{2}N_T$. Numerical simulations show that $\bar{S}$ inverts slightly faster than S from section 1.5.2, with greater advantage on finer grids. It seems possible that $\bar{S}$ may have better conditioning than S, whose condition number behaves as $O\left(\frac{1}{h^2}\right)$; this may be due to the fact that the entries of $\bar{S}$ depend on h as $O(1)$. This conjecture is inspired by our computational experience; its rigorous investigation is beyond the scope of the current study.

1.5.4 Solenoidal approach

In this section we construct an explicit basis of $\mathbf{V}_h$ and use it to solve (1.21) directly. Divergence-free elements were first constructed by Crouzeix [14] and Thomasset [85]. See also [17, p. 295] for a more general derivation.

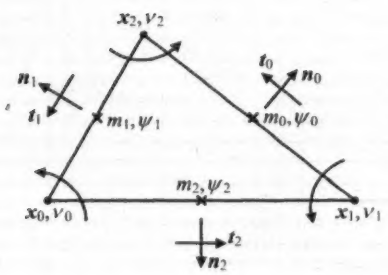

Figure 2: Triangular element with Crouzeix-Raviart and divergence-free degrees of freedom.

Local basis. Let us first consider one element $\tau \in \mathcal{T}_h$ – all indices used in this paragraph are local to τ (Figure 2.) We have six Crouzeix-Raviart basis functions – three tangential and three normal:

$$Y = \Big[\mathbf{t}_0\psi_0, \mathbf{t}_1\psi_1, \mathbf{t}_2\psi_2, \mathbf{n}_0\psi_0, \mathbf{n}_1\psi_1, \mathbf{n}_2\psi_2\Big]. \tag{1.70}$$

We seek functions $\boldsymbol{\nu}$, which are their linear combinations, i.e.

$$\boldsymbol{\nu} = \sum_{i=0}^{2} \big(a_i\mathbf{t}_i\psi_i + b_i\mathbf{n}_i\psi_i\big), \tag{1.71}$$

and satisfy the incompressibility constraint

$$\int_\tau \nabla\cdot\boldsymbol{\nu}\ dx = 0. \tag{1.72}$$

Substitute (1.71) into (1.72) and calculate

$$\begin{aligned}
\int_\tau \nabla\cdot\boldsymbol{\nu}\ dx &= \sum_{i=0}^{2}\left[a_i\int_\tau \nabla\cdot(\mathbf{t}_i\psi_i)\ dx + b_i\int_\tau \nabla\cdot(\mathbf{n}_i\psi_i)\ dx\right] \\
&= \sum_{i=0}^{2}\left[a_i\int_{\partial\tau}(\mathbf{t}_i\psi_i)\cdot\mathbf{n}\ ds + b_i\int_{\partial\tau}(\mathbf{n}_i\psi_i)\cdot\mathbf{n}\ ds\right] \\
&= \sum_{i=0}^{2}\sum_{j=0}^{2}\left[a_i\int_{f_j}\psi_i(\mathbf{t}_i\cdot\mathbf{n}_j)\ ds + b_i\int_{f_j}\psi_i(\mathbf{n}_i\cdot\mathbf{n}_j)\ ds\right]
\end{aligned}$$

$$= \sum_{i=0}^{2} \sum_{j=0}^{2} \left[a_i (\mathbf{t}_i \cdot \mathbf{n}_j) \underbrace{\int_{f_j} \psi_i \, ds}_{=|f_i|\delta_{ij}} + b_i (\mathbf{n}_i \cdot \mathbf{n}_j) \underbrace{\int_{f_j} \psi_i \, ds}_{=|f_i|\delta_{ij}} \right]$$

$$= \sum_{i=0}^{2} \left[a_i \underbrace{(\mathbf{t}_i \cdot \mathbf{n}_i)}_{=0} |f_i| + b_i \underbrace{(\mathbf{n}_i \cdot \mathbf{n}_i)}_{=1} |f_i| \right]$$

$$= \sum_{i=0}^{2} \left[a_i 0 + b_i |f_i| \right].$$

Here $\{f_i\}_{i=0}^2$ are the faces of τ, and $|f_j|$ denotes the length of f_j. Therefore, the matrix of the divergence operator in span Y in the basis (1.70) is a 1×6 matrix given by

$$D_\tau = \left[0, 0, 0, |f_0|, |f_1|, |f_2| \right]. \tag{1.73}$$

The vector of coefficients

$$\left[a_0, a_1, a_2, b_0, b_1, b_2 \right]$$

of every divergence-free function $\boldsymbol{\nu}$ belongs to the kernel of D_τ. It is not hard to establish that $\ker D_\tau$ is spanned by the following vectors:

$$\begin{aligned}
&\left[1, 0, 0, 0, 0, 0 \right], \\
&\left[0, 1, 0, 0, 0, 0 \right], \\
&\left[0, 0, 1, 0, 0, 0 \right], \\
&\left[0, 0, 0, 0, \frac{1}{|f_1|}, \frac{-1}{|f_2|} \right], \\
&\left[0, 0, 0, \frac{-1}{|f_0|}, 0, \frac{1}{|f_2|} \right], \\
&\left[0, 0, 0, \frac{1}{|f_0|}, \frac{-1}{|f_1|}, 0 \right].
\end{aligned} \tag{1.74}$$

The first three vectors give the tangential basis functions, which are divergence-free and will be used in the basis of $\mathbf{V}_h$. We denote

$$\boldsymbol{\nu}_{t,i} = \mathbf{t}_i \psi_i, \qquad i \in \{0, 1, 2\}. \tag{1.75}$$

Of course, $\ker D_\tau$ has dimension 5 and we can see clearly that the last three vectors in (1.74) are linearly dependent. They give us the following

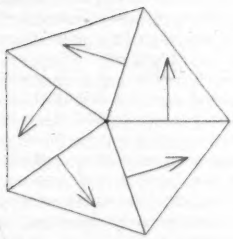

Figure 3: Two dimensional divergence-free basis function associated with a node.

divergence-free functions

$$\begin{aligned}
\boldsymbol{\nu}_{n,0} &= \frac{\mathbf{n}_1}{|f_1|}\psi_1 + \frac{-\mathbf{n}_2}{|f_2|}\psi_2, \\
\boldsymbol{\nu}_{n,1} &= \frac{\mathbf{n}_2}{|f_2|}\psi_2 + \frac{-\mathbf{n}_0}{|f_0|}\psi_0, \\
\boldsymbol{\nu}_{n,2} &= \frac{\mathbf{n}_0}{|f_0|}\psi_0 + \frac{-\mathbf{n}_1}{|f_1|}\psi_1,
\end{aligned} \tag{1.76}$$

any two of which are linearly independent.

Notice in (1.76) that the normal basis functions are combined in a way which ensures that the influx through one face is exactly canceled by the outflux through another face, guaranteeing mass conservation. Also, we have assigned indices to $\boldsymbol{\nu}_{n,i}$ deliberately so that the function $\boldsymbol{\nu}_{n,i}$ has, in its definition above, the two faces containing node i, and the signs are such that the normals go counter-clockwise around the node.

Global basis. The six divergence-free functions in (1.75) and (1.76) are the restrictions to τ of the global divergence-free basis functions, which we will define next. It is clear that the tangential functions in the Crouzeix-Raviart basis of $\mathbf{Y}_h$ can also be used in the basis of $\mathbf{V}_h$:

$$\boldsymbol{\nu}_{t,i} = \mathbf{t}_i\psi_i, \qquad 0 \le i < N_F. \tag{1.77}$$

Here we work on the whole grid and the indices are global.

What remains, is to construct the rest of the basis functions of $\mathbf{V}_h$ as linear combinations of the normal Crouzeix-Raviart basis functions. Following our insight from the local basis, we will construct one function for each grid node (see Figure 3):

$$\boldsymbol{\nu}_{n,j} = \sum_{i=0}^{N_F-1} \frac{\varepsilon_{i,j}\mathbf{n}_i}{|f_i|}\psi_i, \qquad 0 \le j < N_N, \tag{1.78}$$

where

$$\varepsilon_{i,j} = \begin{cases} 0, & \text{if face } i \text{ does not contain node } j, \\ 1, & \text{if } \mathbf{n}_i \text{ goes counter-clockwise around node } j, \\ -1, & \text{if } \mathbf{n}_i \text{ goes clockwise around node } j. \end{cases} \tag{1.79}$$

It is trivial to verify that the restriction of any of the functions defined in (1.78) to an element is either 0 or one of the functions in (1.76), which is sufficient to conclude that $\boldsymbol{\nu}_{n,j} \in \mathbf{V}_h$ for all $j \in \{0, \dots N_N - 1\}$. Also, we calculate the dimension of $\mathbf{V}_h$ as the dimension of $\mathbf{Y}_h$ minus the number of constraints. Taking into account Euler's formula relating the numbers of nodes, faces, and elements in a grid ($N_N - N_F + N_T = 1$), we derive

$$\dim \mathbf{V}_h = \dim \mathbf{Y}_h - N_T = 2N_F - N_T = N_F + N_N - 1. \tag{1.80}$$

Therefore, $\boldsymbol{\nu}_{t,i}$ and $\boldsymbol{\nu}_{n,j}$ are one too many. It is clear that the tangential components are linearly independent, thus the normal divergence-free functions must be linearly dependent. If we remove any one of them, the rest are linearly independent and, together with the tangential components, form a basis of $\mathbf{V}_h$. A proof of this fact can be found in [17, p. 307].

Again, the tangential components are decoupled and left unchanged by the projection. The matrix of the system for the normal components can be assembled in the usual way from local mass matrices of the divergence-free basis. To obtain a local matrix, we multiply the standard Crouzeix-Raviart local mass matrix on the left and on the right by the local coefficients matrix, which has rows equal to the vectors in (1.74):

$$M_\tau^{\nu_n} = N_\tau M_\tau N_\tau^T, \tag{1.81}$$

where

$$N_\tau = \begin{pmatrix} 0 & \frac{\varepsilon_{0,1}}{|f_1|} & \frac{\varepsilon_{0,2}}{|f_2|} \\ \frac{\varepsilon_{1,0}}{|f_0|} & 0 & \frac{\varepsilon_{1,2}}{|f_2|} \\ \frac{\varepsilon_{2,0}}{|f_0|} & \frac{\varepsilon_{2,1}}{|f_1|} & 0 \end{pmatrix}. \tag{1.82}$$

Boundary conditions on the normal components are prescribed by the following iterative procedure. We start by choosing one boundary node and prescribing value 0 for the coefficient of its $\boldsymbol{\nu}_{n,i}$. This is how we remove the linearly dependent function. Then we traverse the boundary faces going in

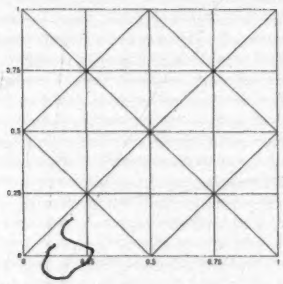

Figure 4: Sample 4×4 grid.

one direction around Ω. One of the nodes on the current face will have its coefficient already assigned. The other one we compute so that the normal boundary condition on the face is satisfied. It is easy to see that if the boundary condition satisfies (1.55), then, when we get back to the starting node, we will get a value equal to 0. Unfortunately, it is not possible to impose boundary conditions, if the domain is not simply connected, or if there are parts of $\partial\Omega$ with Dirichlet boundary conditions separated by boundary conditions of different types.

The size of the matrix that we need to invert for the projection is equal to the number of interior nodes, which is by far the smallest of the three suggested implementations. Moreover, it is also the fastest to resolve, significantly improving the time of the inter-elemental multipliers, as well as the time of the elemental multipliers. The solenoidal approach is our method of choice, with the only limitation possibly coming from the boundary conditions.

1.5.5 Numerical results

The performance of the proposed scheme was evaluated with a series of numerical experiments. All tests discussed in this section, unless otherwise specified, were performed using the first order in time pressure-correction scheme. The grids we used were produced by subdividing the domain into squares and dividing each square into eight triangles using the two diagonals and two lines connecting the midpoints of opposite sides of the squares. A sample 4×4 grid is illustrated in Figure 4.

We start with a comparison of the three implementations of the projection step. The test example is the analytic solution discussed in the next paragraph, solved for 100 timesteps with $\Delta t = 0.01$. The computations were performed without advection (time-dependent Stokes problem), since we are interested only in the performance of the projection step, and the advection does not play a role in it. We ran the same test using Lagrange multipliers,

Table 1: Comparison of three implementations of the projection step in two dimensions. CPU times are given in seconds.

	LM			IM			DF		
grid	dim	iters	CPU	dim	iter	CPU	dim	iter	CPU
16×16	511	180	$\leq$0.001	736	95	$\leq$0.001	225	40	$\leq$0.001
32×32	2047	360	0.03	3008	195	0.03	961	80	$\leq$0.001
64×64	8191	720	0.39	12160	390	0.34	3969	158	0.03
128×128	32767	1410	4.4	48896	775	4.6	16129	310	0.36
256×256	131071	2800	43	196096	1550	37	65025	610	4.7

LM – Lagrange multipliers (1.5.2)
IM – Inter-elemental multipliers (1.5.3)
DF – divergence-free basis (1.5.4)

inter-elemental multipliers, and divergence-free basis on five different grids and recorded the average number of iterations and the average CPU time needed for the projection step in each case. In all cases the linear system was inverted by a conjugate gradient method. Table 1 outlines the results. We see that the inter-elemental multipliers give slightly improved times as the number of iterations is about half of that of the Lagrange multipliers, while the number of unknowns approaches 1.5 times the number of Lagrange multipliers as the grid gets finer. Using the divergence-free basis, on the other hand, gives both the smallest system (dimension is less than half of that of Lagrange multipliers) and the smallest number of iterations (around 40% of that of inter-elemental multipliers), which ultimately leads to tremendous reduction in computational times. In view of these results, all remaining tests in this section were performed using the divergence-free basis.

Convergence tests. We confirmed the convergence of the scheme by running a convergence test using an analytic solution given by

$$u = \sin x \sin(y+t), \quad v = \cos x \cos(y+t), \quad p = \cos x \sin(y+t). \tag{1.83}$$

The problem was solved for $t \in [0,5]$ in the square $[0,1] \times [0,1]$ with initial and boundary conditions given by the exact solution, and a source term prescribed to satisfy the incompressible Navier-Stokes equations at Reynolds number $Re = 100$. Figure 5 presents the error of the computed velocity in $\ell^2(L^2)$-norm plotted versus the timestep for a variety of grids. The graph indicates first-order accuracy in time. The flattening observed at small timesteps is due to the saturation of the spatial error. The graphs in Figure 6 present the error in the velocity in $\ell^2(L^2)$- and the $\ell^2(H^1)$-norms versus h for a variety of timesteps. As anticipated, the error in the velocity is second-order in space in L^2-norm and first-order in H^1-norm.

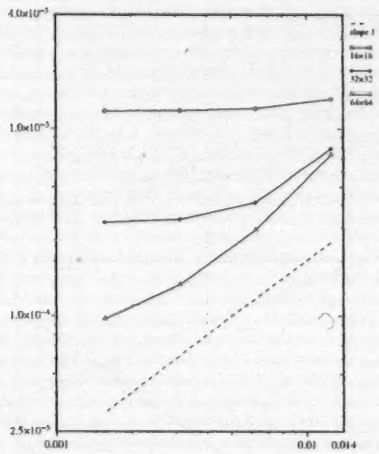

Figure 5: Error in velocity in $\ell^2(L^2)$-norm versus the time step Δt, $T = 5$, $Re = 100$, for a variety of grid sizes.

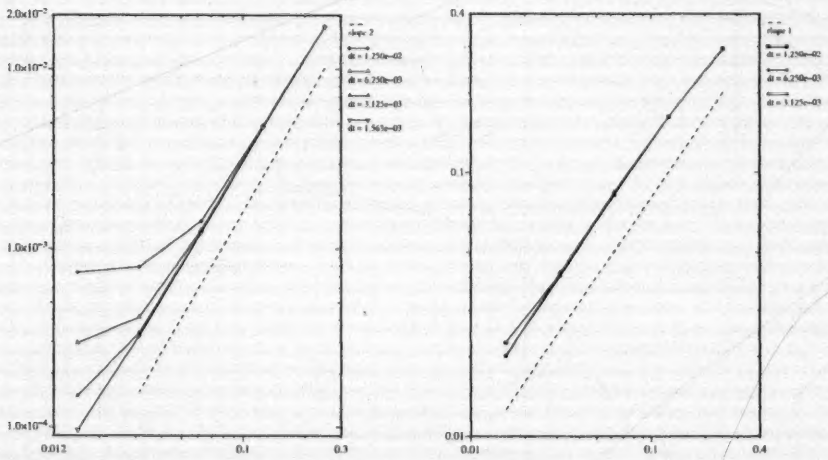

Figure 6: Error in velocity in $\ell^2(L^2)$-norm (left) and $\ell^2(H^1)$-norm (right) versus the grid size h, $T = 5$, $Re = 100$, for a variety of time steps.

Table 2: Comparison of the $\ell^2(\mathbf{L}^2)$ and $\ell^2(\mathbf{H}^1)$-norms of the velocity error for different time steps, Δt, and different meshsize h; Reynolds number $Re = 100$.

		$\ell^2(\mathbf{L}^2)$ norm		$\ell^2(\mathbf{H}^1)$-norm	
Δt	grid	$iso-\mathbb{P}_2\mathbb{P}_1$	Present scheme	$iso-\mathbb{P}_2\mathbb{P}_1$	Present scheme
0.0500000	8×8	1.702290e-02	1.0051450e-02	3.850517e-01	1.8557660e-01
0.0250000	8×8	1.623382e-02	5.3463740e-03	3.818002e-01	1.6369440e-01
0.0125000	8×8	1.617754e-02	4.8185810e-03	3.815409e-01	1.6200860e-01
0.0062500	8×8	1.618194e-02	4.7534930e-03	3.816086e-01	1.6191400e-01
0.0031250	8×8	1.618895e-02	4.7361680e-03	3.816846e-01	1.6186320e-01
0.0015625	8×8	1.619345e-02	4.7284610e-03	3.817329e-01	1.6181490e-01
0.0500000	16×16	5.855242e-03	8.8821840e-03	1.540878e-01	1.2183130e-01
0.0250000	16×16	3.367833e-03	2.6469600e-03	1.435928e-01	8.5758860e-02
0.0125000	16×16	3.046877e-03	1.4165660e-03	1.423505e-01	8.2413190e-02
0.0062500	16×16	3.003900e-03	1.2606420e-03	1.421816e-01	8.3748370e-02
0.0031250	16×16	2.998113e-03	1.2336630e-03	1.421746e-01	8.4228420e-02
0.0015625	16×16	2.998189e-03	1.2257410e-03	1.421884e-01	8.4313620e-02
0.0500000	32×32	4.905032e-03	8.8466680e-03	8.323091e-02	1.0610640e-01
0.0250000	32×32	1.579660e-03	2.3649390e-03	5.455816e-02	5.0030510e-02
0.0125000	32×32	7.565248e-04	7.8401870e-04	5.085209e-02	4.0976360e-02
0.0062500	32×32	5.509742e-04	4.0827610e-04	5.028274e-02	4.0347290e-02
0.0031250	32×32	4.993708e-04	3.3217460e-04	5.016987e-02	4.1510130e-02
0.0015625	32×32	4.863987e-04	3.1999660e-04	5.014688e-02	4.2656160e-02

Comparison with other elements. We compared the accuracy of the proposed method with the accuracy of another inf-sup stable element featuring linear velocity – the $iso-\mathbb{P}_2\mathbb{P}_1$ (Bercovier-Pironneau) element. The computational meshes contain the same nodes, faces and elements for the velocity for both schemes, while the pressure representation is different due to the specific requirements of the Bercovier-Pironneau element. The errors in the predicted velocities for both schemes are listed in Table 2; they indicate that in both cases the convergence is the expected $O(h^2)$ in L^2-norm and $O(h)$ in H^1-norm.

We also compared with the accuracy of the pressure-correction scheme implemented using Crouzeix-Raviart elements for both the advection-diffusion and the projection steps. Because the Crouzeix-Raviart velocity is generally discontinuous, the advection must be treated in the spirit of discontinuous Galerkin methods and will depend on the choice of interfacial fluxes. For the sake of fair comparison, we omitted the advection terms from both schemes and compared their performance on the generalized Stokes problem. Figure 7 presents the errors in velocity depending on the grid size h and the timestep Δt; we see that the two schemes give nearly identical results. Figure 8 shows the convergence of the velocities in time when a second-order-in-time backward difference scheme was used. In both L^2- and H^1-norms the convergence is clearly second order in time.

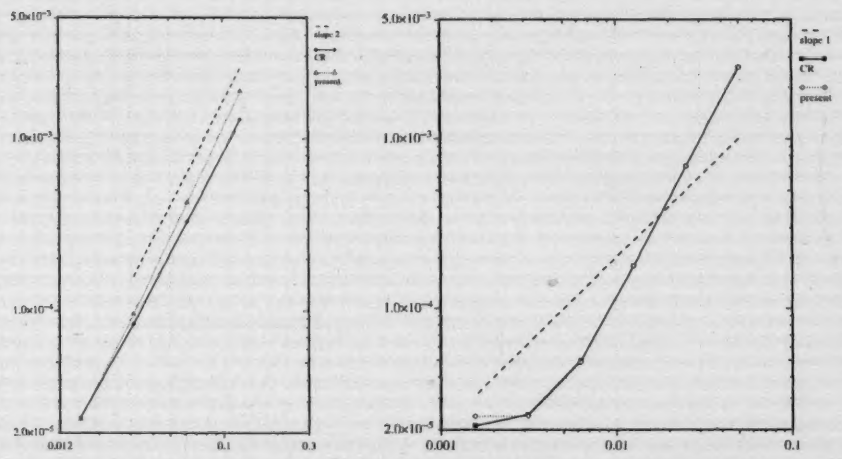

Figure 7: Error in velocity in $\ell^2(L^2)$-norm versus the grid size h at $\Delta t = 0.003125$ (left), and versus the time step Δt at $h = 1/64$ (right); $T = 5$, $Re = 0$. The results are produced with an incremental scheme using Crouzeix-Raviart elements for both the momentum equation and the projection (CR), and using the present scheme (present).

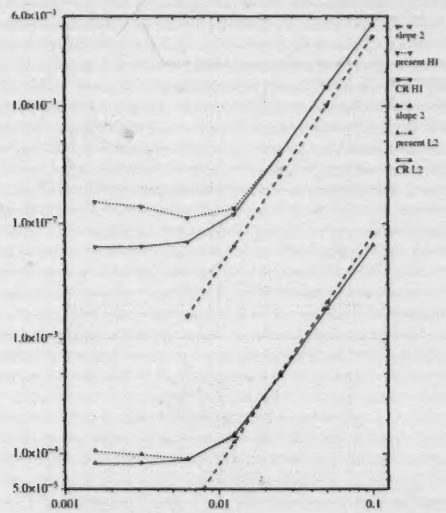

Figure 8: Error in velocity in $\ell^2(L^2)$-norm and $\ell^2(H^1)$-norm versus the time step Δt at $h = 1/32$; $T = 5$, $Re = 0$. The results are produced with a second-order-in-time incremental scheme using Crouzeix-Raviart elements for both, the momentum equation and the projection (CR) and using the present scheme (present).

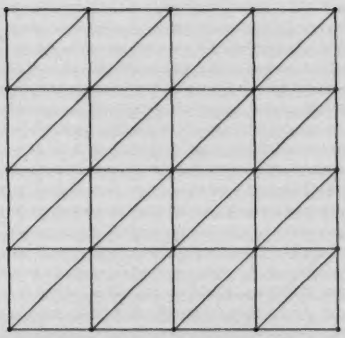

Figure 9: Sample 4×4 diagonal grid.

Table 3: The $\ell^2(L^2)$ norm of the velocity error using a diagonal grid, different time steps, Δt, and a different meshsizes h; Reynolds number $Re = 100$.

$\Delta t \backslash h$	8×8	16×16	32×32	64×64	128×128
0.0500000	1.07102e-02	1.002422e-02	1.015209e-02	1.027519e-02	1.033813e-02
0.0250000	5.23603e-03	2.801070e-03	2.643430e-03	2.652358e-03	2.665173e-03
0.0125000	4.61674e-03	1.296322e-03	8.219813e-04	7.832054e-04	7.826101e-04
0.0062500	4.47716e-03	1.177908e-03	3.887040e-04	2.988708e-04	2.903357e-04
0.0031250	4.33028e-03	1.416590e-03	3.177499e-04	1.498433e-04	1.324947e-04
0.0015625	3.68170e-03	1.641200e-03	5.048719e-04	1.043321e-04	6.791169e-05

Diagonal grids. Without the existence of a divergence-free velocity interpolant in $\mathbb{P}_1$, we were able to establish only suboptimal convergence rate for the velocity. The so-called diagonal grid (Figure 9) is an example of a problematic grid, where the $\mathbb{P}_1\mathbb{P}_0$ element is known to lock, in the sense that the only divergence-free velocity is the constant 0 (assuming homogeneous Dirichlet boundary conditions.) This happens when there is an internal node directly connected to more than two boundary nodes. Then there are at least two elements containing this node with their other two nodes on the boundary. Imposing velocity divergence equal to zero in these elements uniquely determines the values of the velocity at the node to equal 0. This effect then propagates throughout the whole grid, giving constant 0 as the only divergence-free velocity on such grid. In the suggested scheme this negative effect is avoided, since the $\mathbb{P}_1$ velocity is never forced to be incompressible.

Another perspective on the issue of locking is to consider a steady solution. Let $\tilde{\mathbf{u}}_h$ and $\mathbf{u}_h$ be the limits of $\tilde{\mathbf{u}}_h^n$ and $\mathbf{u}_h^n$ respectively as n approaches infinity. If we are on a problematic grid, there exists a spurious pressure gradient $\mathbf{g}_h = \mathbf{u}_h - \tilde{\mathbf{u}}_h \in \mathbf{Y}_h$ such that $(\mathbf{g}_h, \tilde{\mathbf{v}}_h) = 0$ for all $\tilde{\mathbf{v}}_h \in \mathbf{X}_h$. Since the inf-sup condition between $\mathbf{X}_h$ and Q_h is not satisfied, nothing is pressing $\mathbf{g}_h$ to

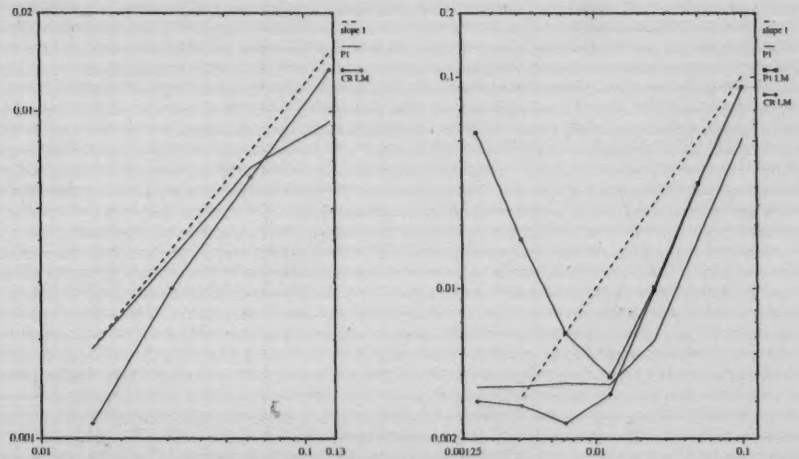

Figure 10: Error in the pressure in $\ell^2(L^2)$-norm versus the grid size h at $\Delta t = 0.003125$ (left), and versus the time step Δt on grid 32×32 (right).

P1 LM – $\mathbb{P}_0$ pressure derived from Lagrange multipliers
P1 – $\mathbb{P}_1$ pressure recovered by means of (1.25)
CR LM – $\mathbb{P}_0$ pressure computed using classical Crouzeix-Raviart elements

vanish as the velocity approximations approach the steady solution. Therefore, enforcing $\mathbf{u}_h$ to be locally solenoidal does not automatically require the same of $\tilde{\mathbf{u}}_h$, thereby avoiding the locking effect.

Table 3 shows the results. The scheme has an optimal rate of convergence on a diagonal grid for the test with an analytic solution, however, for tests leading to steady solutions, it was observed that the convergence on a cross-grid was much faster than that on the problematic diagonal grids.

Pressure recovery. In the last test with the analytic solution presented here, we investigated the error in the pressure. We made three series of computations. In the first one, we computed a $\mathbb{P}_0$ approximation of the pressure derived from the Lagrange multipliers of the projection step of the current method. In the second test, we recovered $\mathbb{P}_1$ pressure in postprocessing by means of (1.25). In the third test, we took the $\mathbb{P}_0$ pressure resulting from the use of Crouzeix-Raviart elements in both the viscous and the projection step. Figure 10 presents the errors of the three pressures in $\ell^2(L^2)$-norm versus the mesh size h and the timestep Δt. The pressure derived from the Lagrange multipliers of the current scheme diverges at small Δt, while the pressure associated with the Crouzeix-Raviart elements converges. The $\mathbb{P}_1$ pressure retrieved by (1.25) also converges.

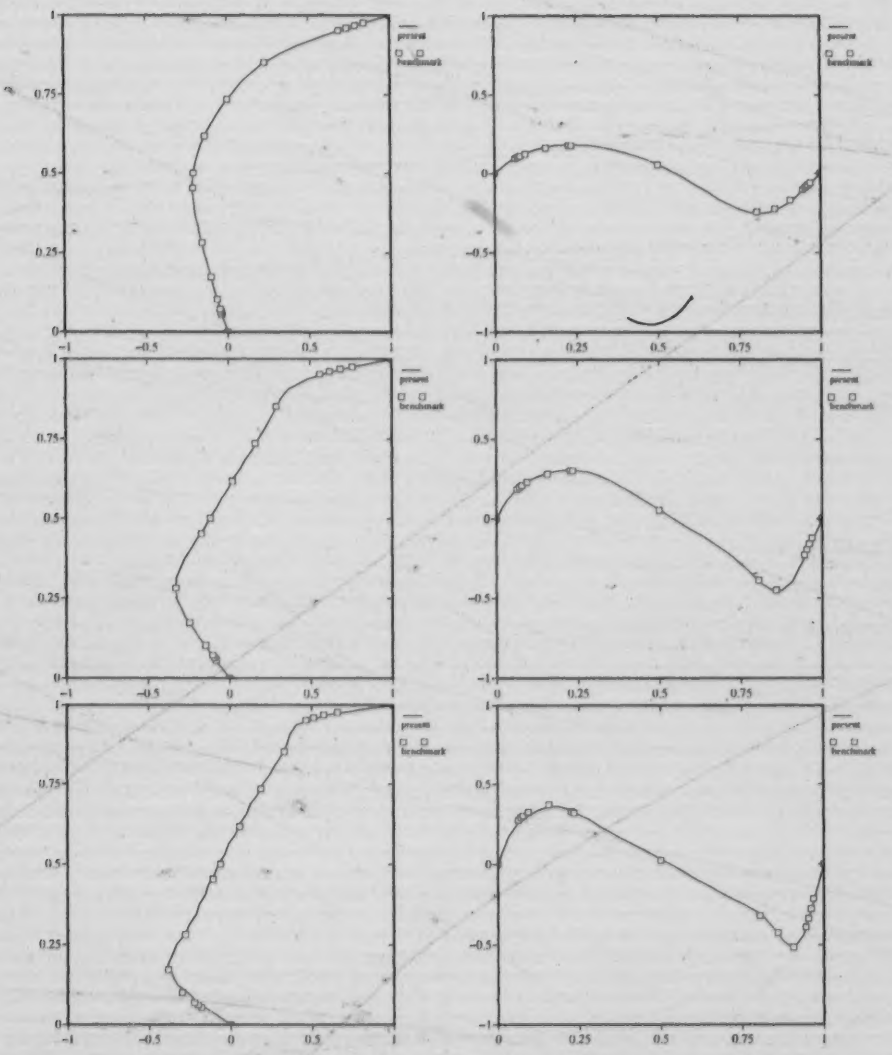

Figure 11: Lid-driven cavity flow at $Re = 100, 400, 1000$ (from top to bottom). Horizontal (left) and vertical (right) velocity profiles through the geometric center of the cavity.

Lid-driven cavity. The 2-dimensional lid-driven cavity problem is a real-world-like problem commonly used for validation of numerical solutions of the Navier-Stokes problem and is documented widely in the literature. It consists of a square cavity in which a viscous fluid is driven by the top, where a boundary condition of $(1,0)$ for the velocity is prescribed. We computed the steady solution on a 32×32 grid using the present scheme for a variety of Reynolds numbers. The results were then compared to the solutions described in [29], where the authors employed a vorticity-stream function finite difference scheme with a multigrid relaxation, and solved the steady equations directly on a 128×128 grid. Figure 11 shows that the two sets of data practically coincide, with the differences being smaller than 10^{-4} in L^∞-norm.

1.6 Implementation in three spatial dimensions

Now we present the case of $N = 3$.

1.6.1 General remarks

A three-dimensional grid consists of nodes, edges, faces, and elements: $\mathcal{G}_h = (\mathcal{N}_h, \mathcal{E}_h, \mathcal{F}_h, \mathcal{T}_h)$, with $|\mathcal{E}_h| = N_E$.

The nodes have three coordinates and are, as before, numbered so that the boundary nodes have the highest indices, to facilitate imposition of Dirichlet boundary conditions.

The edges consist of two nodes, whose indices are taken in increasing order. Each edge $e \in \mathcal{E}_h$ has a unit tangent vector $\mathbf{t}_e$ associated with it, which is taken in the direction from node 0 to node 1.

The faces consist of three nodes taken in no particular order and three edges, indexed so that the edge with the local index i is opposite of the local node i. Faces have three unit vectors associated with them – two tangential and one normal, which we will denote $\mathbf{t}_i, \mathbf{s}_i$, and $\mathbf{n}_i$ for face $f_i \in \mathcal{F}_h$. The first tangent, $\mathbf{t}_i$, is taken to equal the tangent vector associated with the face's local edge 0. The normal vector, $\mathbf{n}_i$, is computed as the cross product of the vector going from local node 0 to local node 1 and the vector going from local node 0 to local node 2, then normalized to unit length. This way, if we look at the face so that the normal vector is pointing towards us, we will see the local indices of the nodes going counter-clockwise around the face. The second tangent vector is computed so that the triple $(\mathbf{t}_i, \mathbf{s}_i, \mathbf{n}_i)$ forms an orthonormal basis with positive orientation, i.e. $\mathbf{s}_i = \mathbf{n}_i \times \mathbf{t}_i$.

The elements consist of four nodes, six edges and four faces. The nodes are numbered so that the scalar triple product of the vectors from local node 0 to local nodes 1, 2, and 3, taken in this order, is positive. The local faces are

again numbered so that the local face i is opposite the local node i. The local edges are numbered in increasing order of the local indices of their end-nodes.

Unlike in the two-dimensional case, here we allow elements to have all four nodes on the boundary of the domain. Otherwise such a requirement would be too restrictive on the grids that we can use. A consequence of dropping this requirement is that now the faces with all (three) nodes on the boundary are not necessarily boundary faces, Therefore, simply sorting faces in increasing order of the global indices of their nodes, as before, generally does not give us order, in which boundary faces have highest indices. For this reason, we discriminate among faces according to the number of elements containing them – two for interior faces and one for boundary faces; then we sort them so that boundary faces are numbered last. An edge is on the boundary if it belongs to a boundary face. The edges are also ordered so that boundary edges are last, although we will impose an additional requirement on the order of edges, which we will discuss later.

Deciding whether the normal vector of a face is inner or outer to one of its elements can be done by the following procedure. First construct an even permutation of $\{0, 1, 2, 3\}$ in which the local index of the face is first and the local index of the node, which is the face's local node 0, is second. Then we compare the global index of the element node, which is third in the permutation, to the global index of the face's node 1. If they are equal, then the normal is outer to the element, otherwise it is inner. This procedure is justified by the rules we have for the order of nodes in elements and the direction of faces' normal vectors; it can be proved by direct verification as there is only a finite and small number of possibilities.

Another significant difference with the two-dimensional case is the fact that the local mass matrix of the Crouzeix-Raviart element is not diagonal. Unfortunately, a Gaussian quadrature formula featuring the four centroids of the faces as integration nodes is not exact for integrating quadratic functions over a tetrahedron. This quadrature is, however, exact for linear functions, i.e. it does give second order of accuracy. Therefore, if we use this formula instead of exact integration to compute the entries of the local mass matrix, we will end up with a lumped diagonal matrix, whose entries are $O(h^2)$ approximations of the entries of the exact mass matrix.

We will use the same notation for different basis functions and expansion coefficients as in the two-dimensional case (section 1.5.1).

1.6.2 Divergence-free basis

Encouraged by the success of the solenoidal approach in two dimensions, we start our three-dimensional implementation by constructing a divergence-free

basis of $\mathbf{V}_h$. Such construction was first done by Hecht in [41]. We also refer the reader to [17, p.312], where the same construction is repeated, and [69], where divergence-free basis is constructed for the three-dimensional Raviart-Thomas-Nédélec elements applied to Darcy problems.

Local basis. The derivation of the three-dimensional local divergence-free basis is exactly the same as in the two-dimensional case (see section 1.5.4). We will omit most derivation details and present thoroughly only the end results.

There are twelve local Crouzeix-Raviart basis functions in an element $\tau \in \mathcal{T}_h$. In tangential-normal components we write them this way:

$$Y = \Big[\mathbf{t}_0\psi_0, \mathbf{t}_1\psi_1, \mathbf{t}_2\psi_2, \mathbf{t}_3\psi_3, \mathbf{s}_0\psi_0, \mathbf{s}_1\psi_1, \mathbf{s}_2\psi_2, \mathbf{s}_3\psi_3, \mathbf{n}_0\psi_0, \mathbf{n}_1\psi_1, \mathbf{n}_2\psi_2, \mathbf{n}_3\psi_3\Big]. \quad (1.84)$$

In this paragraph the normals $\mathbf{n}_i, i \in \{0,1,2,3\}$ are outer to the element, and the tangents are such that the triple $(\mathbf{t}_i, \mathbf{s}_i, \mathbf{n}_i)$ is orthonormal and with positive orientation. We seek coefficients vectors

$$\Big[a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, c_0, c_1, c_2, c_3\Big]$$

such that the function

$$\boldsymbol{\nu} = \sum_{i=0}^{3} \left(a_i\mathbf{t}_i\psi_i + b_i\mathbf{s}_i\psi_i + c_i\mathbf{n}_i\psi_i\right) \quad (1.85)$$

is divergence-free. The matrix of the local discrete divergence operator, expressed in the basis (1.84), is

$$D_\tau = \Big[0,0,0,0,0,0,0,0,|f_0|,|f_1|,|f_2|,|f_3|\Big]. \quad (1.86)$$

Again we see that the eight tangential functions are divergence-free, so we set

$$\begin{aligned} \boldsymbol{\nu}_{t,i} &= \mathbf{t}_i\psi_i, \\ \boldsymbol{\nu}_{s,i} &= \mathbf{s}_i\psi_i, \end{aligned} \qquad i \in \{0,1,2,3\}. \quad (1.87)$$

Other than the tangential components, we have the following six coefficient

vectors in the kernel of D_τ:

$$\begin{aligned}
&\left[0,0,0,0,0,0,0,0,0,0,\frac{1}{|f_2|},\frac{-1}{|f_3|}\right], \quad \left[0,0,0,0,0,0,0,0,0,0,\frac{-1}{|f_1|},0,\frac{1}{|f_3|}\right],\\
&\left[0,0,0,0,0,0,0,0,0,0,\frac{1}{|f_1|},\frac{-1}{|f_2|},0\right], \quad \left[0,0,0,0,0,0,0,0,0,\frac{1}{|f_0|},0,0,\frac{-1}{|f_3|}\right], \quad (1.88)\\
&\left[0,0,0,0,0,0,0,0,\frac{-1}{|f_0|},0,\frac{1}{|f_2|},0\right], \quad \left[0,0,0,0,0,0,0,0,\frac{1}{|f_0|},\frac{-1}{|f_1|},0,0\right].
\end{aligned}$$

The dimension of $\ker D_\tau$ is 11. It is easy to see that any three vectors in (1.88) are linearly independent, as long as they contain at least one nonzero coefficient in each of the last four columns.

The divergence-free functions are

$$\begin{aligned}
\boldsymbol{\nu}_{n,0} &= \frac{\mathbf{n}_2}{|f_2|}\psi_2 + \frac{-\mathbf{n}_3}{|f_3|}\psi_3, & \boldsymbol{\nu}_{n,1} &= \frac{\mathbf{n}_3}{|f_3|}\psi_3 + \frac{-\mathbf{n}_1}{|f_1|}\psi_1,\\
\boldsymbol{\nu}_{n,2} &= \frac{\mathbf{n}_1}{|f_1|}\psi_1 + \frac{-\mathbf{n}_2}{|f_2|}\psi_2, & \boldsymbol{\nu}_{n,3} &= \frac{\mathbf{n}_0}{|f_0|}\psi_0 + \frac{-\mathbf{n}_3}{|f_3|}\psi_3, \qquad (1.89)\\
\boldsymbol{\nu}_{n,4} &= \frac{\mathbf{n}_2}{|f_2|}\psi_2 + \frac{-\mathbf{n}_0}{|f_0|}\psi_0, & \boldsymbol{\nu}_{n,5} &= \frac{\mathbf{n}_0}{|f_0|}\psi_0 + \frac{-\mathbf{n}_1}{|f_1|}\psi_1.
\end{aligned}$$

Similarly to the two-dimensional case, we have pairs of normal basis functions combined with coefficients ensuring that in-flux through one face is balanced by out-flux through another face. This time, each divergence-free function is associated with the common edge of its two participating faces. The signs of the coefficients (which determine the directions of the normal vectors) are chosen to follow a simple right-hand rule. If we point the thumb of our right hand in the direction of the common edge, the normal vectors to the faces containing the edge will point in the same direction as the rest of the fingers of our right hand. Here the direction of the edge is taken from the node with smaller local index towards the other node.

Global basis. The tangential Crouzeix-Raviart basis functions are divergence-free and belong to the basis of $\mathbf{V}_h$. We denote them

$$\begin{aligned}
\boldsymbol{\nu}_{t,i} &= \mathbf{t}_i\psi_i,\\
\boldsymbol{\nu}_{s,i} &= \mathbf{s}_i\psi_i,
\end{aligned} \qquad 0 \le i < N_F. \qquad (1.90)$$

We also associate one divergence-free function $\boldsymbol{\nu}_{n,j}$ with each edge $e_j \in \mathcal{E}_h$. For this purpose, we define positive direction of going around the edge using the same right-hand rule as in the local basis, although this time the direction of the edge (and our right-hand thumb) is taken in the direction of the tangent

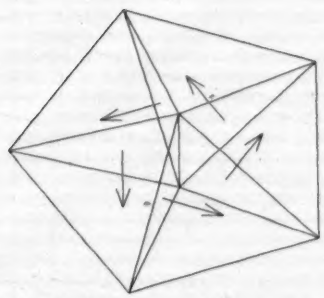

Figure 12: Three-dimensional divergence-free basis function associated with an edge.

vector associated with the edge, as defined in section 1.6.1. Next we define

$$\boldsymbol{\nu}_{n,j} = \sum_{i=0}^{N_F-1} \frac{\varepsilon_{i,j}\mathbf{n}_i}{|f_i|}\psi_i, \qquad 0 \le j < N_E, \tag{1.91}$$

where

$$\varepsilon_{i,j} = \begin{cases} 0, & \text{if face } i \text{ does not contain edge } j, \\ 1, & \text{if } \mathbf{n}_i \text{ goes in positive direction around edge } j, \\ -1, & \text{if } \mathbf{n}_i \text{ goes in negative direction around edge } j. \end{cases} \tag{1.92}$$

An example is given in Figure 12. The functions defined in (1.91) are divergence-free. They are linearly dependent, and in order to complete the basis of $\mathbf{V}_h$, we need to choose a linearly independent subset of them.

The dimension of $\mathbf{V}_h$ is calculated using the Euler's formula $N_N - N_E + N_F - N_T = 1$:

$$\dim \mathbf{V}_h = \dim \mathbf{Y}_h - N_T = 3N_F - N_T = 2N_F + N_E - N_N + 1.$$

$2N_F$ degrees of freedom are accounted for by the tangential functions in (1.90). The remaining $N_E - (N_N - 1)$ functions can be chosen among the normal functions in (1.91). In fact, we will choose the $N_N - 1$ functions, which will not be in the basis. For this purpose we need to construct a *spanning tree* of the grid.

A spanning tree is a graph, whose vertices are grid nodes in $\mathcal{N}_h$, and whose arcs are grid edges in $\mathcal{H}_h \subset \mathcal{E}_h$. In order to be a spanning tree, this graph must be a tree and every node in $\mathcal{N}_h$ must be an endpoint of at least one edge in $\mathcal{H}_h$. Since the edges in $\mathcal{H}_h$ form a tree with N_N vertices, their number is $N_N - 1$,

and this is exactly the number of $\boldsymbol{\nu}_{n,j}$ functions we need to remove. The fact that the rest of the functions are linearly independent is demonstrated in [41].

The construction of the spanning tree can be done by a simple iterative procedure.

- Start with an empty $\mathcal{H}_h$. Choose a starting node in $\mathcal{N}_h$ and mark it as *visited*. Then start a loop over the edges in $\mathcal{E}_h$.
- If both nodes of the current edge are *visited*, or if both nodes are not yet *visited*, then go to the next edge.
- Otherwise, add the current edge to $\mathcal{H}_h$ and mark its not *visited* node as *visited*.
- If all nodes are *visited*, then stop. Otherwise, go to the next edge.

  Note: the algorithm doesn't stop when we process all edges, but only when all nodes are *visited*. If we reach the end of $\mathcal{E}_h$ and we are not done yet, we loop around and go through the edges again. The computational time of this algorithm is $O(N_E)$ (see [1]).

For simplicity of the implementation, we don't actually construct the spanning tree, rather, we just mark the edges that belong in $\mathcal{H}_h$. We treat the divergence-free functions of these edges the same way as we treat homogeneous Dirichlet boundary conditions – we replace their equations with equations containing one on the main diagonal and zero in the right-hand side. Then we avoid solving these equations by sorting the interior edges in $\mathcal{H}_h$ to have the highest indices among the interior edges and the boundary edges in $\mathcal{H}_h$ to have lowest indices among the boundary edges. This way, the interior edges, for which we need to invert a matrix, are grouped in the beginning and the boundary edges, for which we need to impose boundary conditions, are grouped at the end.

Boundary conditions. We denote by $\mathcal{F}_{hb} = \mathcal{F}_h \cap \partial\Omega$ the set of boundary faces, by $\mathcal{E}_{hb} = \mathcal{E}_h \cap \partial\Omega$ the set of boundary edges, by $\mathcal{H}_{hb} = \mathcal{H}_h \cap \mathcal{E}_{hb}$ the set of boundary edges in the spanning tree, and by $\mathcal{E}'_{hb} = \mathcal{E}_{hb} \setminus \mathcal{H}_h$ the remaining boundary edges.

Dirichlet boundary conditions on the $\boldsymbol{\nu}_{n,j}$ functions can be imposed by an iterative procedure. We first assign 0 to the coefficients of all $\boldsymbol{\nu}_{n,j}$ of edges in $\mathcal{H}_{hb}$. Then we go through the boundary faces one by one. If exactly two of the current face's edges have their coefficients already prescribed, then we prescribe the third coefficient so that the value of the normal component of $\mathbf{u}_h$

satisfies the boundary condition. We continue until all faces have their boundary conditions imposed this way. To justify the validity of this procedure, we consider a graph $\mathcal{H}'_{hb}$, which is in a way dual to $\mathcal{H}_{hb}$. This new graph has as its vertices the faces in $\mathcal{F}_{hb}$ and as its arcs the edges in $\mathcal{E}'_{hb}$. We will show that $\mathcal{H}'_{hb}$ is actually a tree. This being the case, the leaves of this tree are faces with two edges in $\mathcal{H}_{hb}$, from which our procedure can start, and then we can go up the branches, all the way to the root of the tree, and prescribe boundary conditions to all faces.

To prove that $\mathcal{H}'_{hb}$ is a tree, we start by pointing out that the spanning tree $\mathcal{H}_h$ must be such that the boundary edges in $\mathcal{H}_h$, belonging to each simply connected part of $\partial\Omega$ on which essential boundary conditions are imposed, forms a tree. This is a necessary and sufficient condition for the $\boldsymbol{\nu}_{n,j}$ functions corresponding to interior edges to form a basis of the interior normal components of functions in $\mathbf{V}_h$ (see [41]). In our case, this means that $\mathcal{H}_{hb}$ is a spanning tree of the triangulation imposed by the grid on $\partial\Omega$. Therefore, the number of edges in $\mathcal{H}_{hb}$ is exactly $N_{N_b} - 1$. All remaining boundary edges are arcs in $\mathcal{H}'_{hb}$. Their number can be found by employing the Euler's formula. On the boundary of Ω we have

$$N_{N_b} - N_{E_b} + N_{F_b} = 2.$$

Therefore,

$$N_{E_b} = \big(N_{N_b} - 1\big) + \big(N_{F_b} - 1\big),$$

which shows that the edges in $\mathcal{H}'_{hb}$ are exactly $N_{F_b} - 1$. Also, the graph $\mathcal{H}'_{hb}$ must be connected, because otherwise, there must be a closed path of edges in $\mathcal{H}_{hb}$ separating disconnected subgraphs of $\mathcal{H}'_{hb}$. This contradicts the fact that $\mathcal{H}_{hb}$ is a tree, i.e. has no cycles. From graph theory we know that every connected graph with arcs that are one less than its vertices is a tree (see [4]), which completes the proof.

Notice that the boundary conditions we need to impose are N_{F_b}, while the boundary edges, whose coefficients we assign according to the boundary condition, are one less: $N_{F_b} - 1$. This means that there is one face whose boundary condition is not imposed. It is easy to see that the normal value of the velocity at this face is such that (1.55) holds.

Structure of the system of equations. The matrix of the system of equations we need to solve is the mass matrix of the divergence-free basis. It can be assembled in the usual way from local mass matrices.

The local mass matrix in x-y-z components is block diagonal with three

one-dimensional mass matrices on the diagonal:

$$M_{\mathtt{xyz}} = \begin{pmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & M \end{pmatrix}. \tag{1.93}$$

The local mass matrix in t-s-n components can be obtained by multiplying on the left and on the right by a transformation matrix:

$$M_{\mathtt{tsn}} = K_{\mathtt{tsn}} M_{\mathtt{xyz}} K_{\mathtt{tsn}}^T. \tag{1.94}$$

The local matrix $K_{\mathtt{tsn}}$ consists of 3 rows and 3 columns of 4×4 diagonal blocks. The block in position (i, j) has on its diagonal the j-th coordinates of the i-th vectors with values of j from 0 to 2 corresponding to x, y, and z respectively, and values of i form 0 to 2 corresponding to vectors $\mathbf{t}, \mathbf{s}$, and $\mathbf{n}$. We can write

$$K_{\mathtt{tsn}} = \begin{pmatrix} T_x & T_y & T_z \\ S_x & S_y & S_z \\ N_x & N_y & N_z \end{pmatrix}, \tag{1.95}$$

where, for example,

$$T_x = \begin{pmatrix} \mathbf{t}_{0,x} & 0 & 0 & 0 \\ 0 & \mathbf{t}_{1,x} & 0 & 0 \\ 0 & 0 & \mathbf{t}_{2,x} & 0 \\ 0 & 0 & 0 & \mathbf{t}_{3,x} \end{pmatrix}, \qquad \text{etc.} \tag{1.96}$$

Here, the normal and the tangential vectors are taken according to the vectors globally assigned to each face in the grid. This must be the case in order to ensure that the local matrices of two elements that share a face will be computed with the same tangential and normal vectors assigned to the common face.

Finally, we compute the local mass matrix of the divergence-free basis by multiplying on the left and on the right by yet another transformation matrix:

$$M_{\mathtt{tsdf}} = K_{\mathtt{tsdf}} M_{\mathtt{tsn}} K_{\mathtt{tsdf}}^T. \tag{1.97}$$

The local matrix $K_{\mathtt{tsdf}}$ has 14 rows and 12 columns, where each row is one of

the coefficient vectors spanning the kernel of D_τ (1.86)

$$K_{\texttt{tsdf}} = \begin{pmatrix} I_{4\times4} & 0_{4\times4} & & 0_{4\times4} & & \\ 0_{4\times4} & I_{4\times4} & & 0_{4\times4} & & \\ & & 0 & 0 & \frac{\varepsilon_{0,2}}{|f_2|} & \frac{\varepsilon_{0,3}}{|f_3|} \\ & & 0 & \frac{\varepsilon_{1,1}}{|f_1|} & 0 & \frac{\varepsilon_{1,3}}{|f_3|} \\ & & 0 & \frac{\varepsilon_{2,1}}{|f_1|} & \frac{\varepsilon_{2,2}}{|f_2|} & 0 \\ 0_{6\times4} & 0_{6\times4} & \frac{\varepsilon_{3,0}}{|f_0|} & 0 & 0 & \frac{\varepsilon_{3,3}}{|f_3|} \\ & & \frac{\varepsilon_{4,0}}{|f_0|} & 0 & \frac{\varepsilon_{4,2}}{|f_2|} & 0 \\ & & \frac{\varepsilon_{5,0}}{|f_0|} & \frac{\varepsilon_{5,1}}{|f_1|} & 0 & 0 \end{pmatrix}. \tag{1.98}$$

After assembling the global matrix, we end up with a system of equations that looks like this:

$$\begin{pmatrix} M_t & M_{ts} & M_{tdf} \\ M_{st} & M_s & M_{sdf} \\ M_{dft} & M_{dfs} & M_{df} \end{pmatrix} \begin{pmatrix} U_t \\ U_s \\ U_{df} \end{pmatrix} = \begin{pmatrix} R_t \\ R_s \\ R_{df} \end{pmatrix}, \tag{1.99}$$

where the right-hand side can be assembled using the global $K_{\texttt{tsdf}}$ and $K_{\texttt{tsn}}$ matrices:

$$\begin{pmatrix} R_t \\ R_s \\ R_{df} \end{pmatrix} = K_{\texttt{tsdf}} K_{\texttt{tsn}} \begin{pmatrix} M\tilde{U}_x \\ M\tilde{U}_y \\ M\tilde{U}_z \end{pmatrix}. \tag{1.100}$$

Note that the global transformation matrices $K_{\texttt{tsdf}}$ and $K_{\texttt{tsn}}$ are not assembled by summing the corresponding local transformation matrices; rather they are assembled by copying entries from the local matrices into the global ones. Let's take an interior face $f \in \mathcal{F}_{hi}$. The local matrices of the two elements sharing this face will have the same entries in rows and columns corresponding to f, since both elements prepare their local matrices using the same globally assigned tangential and normal vectors to f. Thus there will be no contradiction – the entries assigned from different local matrices to the same location in the global matrix will be identical.

The off-diagonal blocks in the last row and the last column of (1.99) are not zero, since the tangential basis functions are not orthogonal to the normal basis functions in the sense of the inner product of $\mathbf{Y}_h$. They were in the two-dimensional case, because the CR mass matrix was diagonal; in the three-dimensional case the tangential and the normal components are fully coupled.

To impose boundary conditions, we replace the equations corresponding to the degrees of freedom on the boundary, and instead of (1.99) we have

$$\begin{pmatrix} M_{t,i} & M_{t,b} & M_{ts,i} & M_{ts,b} & M_{tdf,i} & M_{tdf,b} \\ 0 & I & 0 & 0 & 0 & 0 \\ M_{st,i} & M_{st,b} & M_{s,i} & M_{s,b} & M_{sdf,i} & M_{sdf,b} \\ 0 & 0 & 0 & I & 0 & 0 \\ M_{dft,i} & M_{dft,b} & M_{dfs,i} & M_{dfs,b} & M_{df,i} & M_{df,b} \\ 0 & 0 & 0 & 0 & 0 & I \end{pmatrix} \begin{pmatrix} U_{t,i} \\ U_{t,b} \\ U_{s,i} \\ U_{s,b} \\ U_{df,i} \\ U_{df,b} \end{pmatrix} = \begin{pmatrix} R_{t,i} \\ U_{t,\text{b.cond}} \\ R_{s,i} \\ U_{s,\text{b.cond}} \\ R_{df,i} \\ U_{df,\text{b.cond}} \end{pmatrix} \tag{1.101}$$

This system is not symmetric. We obtain an equivalent symmetric system by removing all $M_{\cdot,b}$ blocks and modifying the right-hand side to reflect the boundary conditions. We actually solve

$$\begin{pmatrix} M_{t,i} & 0 & M_{ts,i} & 0 & M_{tdf,i} & 0 \\ 0 & I & 0 & 0 & 0 & 0 \\ M_{st,i} & 0 & M_{s,i} & 0 & M_{sdf,i} & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ M_{dft,i} & 0 & M_{dfs,i} & 0 & M_{df,i} & 0 \\ 0 & 0 & 0 & 0 & 0 & I \end{pmatrix} \begin{pmatrix} U_{t,i} \\ U_{t,b} \\ U_{s,i} \\ U_{s,b} \\ U_{df,i} \\ U_{df,b} \end{pmatrix} = \begin{pmatrix} \tilde{R}_{t,i} \\ U_{t,\text{b.cond}} \\ \tilde{R}_{s,i} \\ U_{s,\text{b.cond}} \\ \tilde{R}_{df,i} \\ U_{df,\text{b.cond}} \end{pmatrix}, \tag{1.102}$$

where

$$\begin{pmatrix} \tilde{R}_{t,i} \\ \tilde{R}_{s,i} \\ \tilde{R}_{df,i} \end{pmatrix} = \begin{pmatrix} R_{t,i} \\ R_{s,i} \\ R_{df,i} \end{pmatrix} - \begin{pmatrix} M_{t,b} & M_{ts,b} & M_{tdf,b} \\ M_{st,b} & M_{s,b} & M_{sdf,b} \\ M_{dft,b} & M_{dfs,b} & M_{df,b} \end{pmatrix} \begin{pmatrix} U_{t,\text{b.cond}} \\ U_{s,\text{b.cond}} \\ U_{df,\text{b.cond}} \end{pmatrix}. \tag{1.103}$$

It is obvious, and quite disappointing, that this system is very large; it has $2N_F + N_E$ equations and unknowns, of which only $[3N_{F_i} - (N_T - 1)]$ correspond to interior degrees of freedom and actually need to be solved. This system indeed takes excessive time to solve.

1.6.3 Decoupled projection

In an attempt to improve the solution method we, artificially decouple the normal and the tangential components by setting the off-diagonal block matrices in (1.99) to zero. In practice this means that we keep the tangential components of $\tilde{\mathbf{u}}_h$ unchanged and solve the projection problem for the normal

components of $\mathbf{u}_h$ alone:

$$\begin{aligned} &U_t = \tilde{U}_t, \qquad\qquad U_s = \tilde{U}_s, \\ &U_{df,b} = U_{df,\text{b.cond}}, \\ &M_{df,i}U_{df,i} = R_{df,i} - M_{df,b}U_{df,\text{b.cond}}. \end{aligned} \tag{1.104}$$

In the implementation of the velocity-correction scheme, where the right-hand side is available only in a weak form (recall equations (1.26) and (1.29)), we need to invert the CR mass matrix three times to obtain the x, y, and z components of its projection onto $\mathbf{Y}_h$, then extract the normal components and replace them with the solution of the last equation in (1.104). Even with the three inversions of the CR mass matrix, this problem is solved in a small fraction of the time taken by the full solenoidal basis (1.102).

The decoupling suggested here is not exact and introduces additional error in the solution. Numerical experiments show that the approximation we are making is consistent within the overall error of the method. The projected velocity is exactly divergence-free and converges to the exact solution with full order ($O(\Delta t + h^2)$ in $\mathbf{L}^2$-norm for a first order in time scheme). It is even exact for problems where the exact solution is linear. This we explain by noticing that linear velocity is represented exactly in $\mathbf{X}_h$, which means that $\tilde{\mathbf{u}}_h^n$ is already divergence-free and $\mathbf{u}_h^n = \tilde{\mathbf{u}}_h^n$ is the solution of the full projection, as well as the decoupled one. In particular, the tangential components of the two velocities are the same, therefore, the error of the decoupling in this case is zero.

Unfortunately, (1.104) does not resolve the projection exactly when surface tension appears in the right-hand side in the form of a δ-function. One important test for our solver is a problem, in which an initially spherical fluid particle is left to relax in another fluid under the force of surface tension alone. The exact solution of this problem is a constant zero velocity, the shape of the particle remains unchanged, and the pressure is piecewise constant with jump discontinuity across the boundary of the particle, balancing the surface tension.

Constructing a method that is able to resolve this problem exactly is one of the main goals of the current study, and the decoupled projection fails to achieve this. To see why this happens, consider a particle occupying one element. Suppose the pressure is equal to one inside this element and zero everywhere outside. If our method is to be exact, then the discrete gradient of this pressure must exactly balance the hypothetical surface tension acting on the faces of the element. Therefore, if we put the weak gradient of this pressure in the right-hand side of the projection problem, we must get zero velocity as the solution. It takes simple calculus to find that the weak gradient

will have zeros for all tangential components, while the normal components will have nonzero entries only on the faces of the element, where the pressure is discontinuous. These entries will have magnitudes equal to the areas of their corresponding faces and signs giving normals pointing into the element.

First let us see what happens if we are solving the full system. While constructing the right-hand side of the system, we multiply on the left the weak gradient by the global $K_{\texttt{tsdf}}$ matrix giving zero. The solution is obviously zero.

To solve the decoupled system, we invert the CR mass matrix first. The result is devastating – the projection of the weak gradient on $\mathbf{Y}_h$ has nonzero tangential components, which can never be eliminated by projecting only the normal component. Ultimately, the solution is not zero.

The moral of this consideration is that we cannot use the decoupled projection for the solution of free-boundary flows. We can, however, use it as a preconditioner of the matrix of the fully coupled system. Our numerical results show that this preconditioner decreases the computational time to about half of the time needed to solve the system without preconditioner.

1.6.4 Lagrange multipliers

In this section we solve the constrained minimization problem (1.47) again, but this time in three spatial dimensions. As we saw in the previous two sections, the tangential and the normal components are coupled, therefore, we can work in x-y-z coordinates without missing possible simplifications. An equivalent implementation in t-s-n components can be done in a similar manner.

We denote the matrix of the discrete divergence operator in x-y-z by

$$D = \big(D_x, D_y, D_z\big),$$

where D_x, D_y, and D_z are the matrices of the discrete ∂_x, ∂_y, and ∂_z respectively. The linear system resulting from (1.47) is

$$\begin{pmatrix} M & 0 & 0 & D_x^T \\ 0 & M & 0 & D_y^T \\ 0 & 0 & M & D_z^T \\ D_x & D_y & D_z & 0 \end{pmatrix} \begin{pmatrix} U_x \\ U_y \\ U_z \\ \Lambda \end{pmatrix} = \begin{pmatrix} M\tilde{U}_x \\ M\tilde{U}_y \\ M\tilde{U}_z \\ 0 \end{pmatrix}. \tag{1.105}$$

After imposing boundary conditions in the usual way and moving the boundary values to the right-hand side, we get the following system for the interior

degrees of freedom:

$$\begin{pmatrix} M_i & 0 & 0 & D_{x,i}^T \\ 0 & M_i & 0 & D_{y,i}^T \\ 0 & 0 & M_i & D_{z,i}^T \\ D_{x,i} & D_{y,i} & D_{z,i} & 0 \end{pmatrix} \begin{pmatrix} U_{x,i} \\ U_{y,i} \\ U_{z,i} \\ \Lambda \end{pmatrix} = \begin{pmatrix} M_i \tilde{U}_{x,i} \\ M_i \tilde{U}_{y,i} \\ M_i \tilde{U}_{z,i} \\ -D_b U_{\text{b.cond}} \end{pmatrix}, \tag{1.106}$$

where

$$D_b U_{\text{b.cond}} = D_{x,b} U_{x,\text{b.cond}} + D_{y,b} U_{y,\text{b.cond}} + D_{z,b} U_{z,\text{b.cond}}. \tag{1.107}$$

Now we construct the Schur complement of the mass matrix and obtain the following form of the solution:

$$\begin{aligned} &\Lambda = \left(D_{x,i} M_i^{-1} D_{x,i}^T + D_{y,i} M_i^{-1} D_{y,i}^T + D_{z,i} M_i^{-1} D_{z,i}^T\right)^{-1} D\tilde{U}, \\ &U_{\star,i} = \tilde{U}_{\star,i} - M_i^{-1} D_{\star,i}^T \Lambda, \\ &U_{\star,b} = U_{\star,\text{b.cond}}, \\ &\text{where } \star \in \{x, y, z\}. \end{aligned} \tag{1.108}$$

The matrix that needs to be inverted in the equation for Λ above cannot be constructed explicitly, and the only way to solve this problem is via Uzawa type nested iterations. This takes even longer time than inverting the mass matrix of the full divergence-free basis (1.102).

The Uzawa iteration can be avoided and the Schur complement can be constructed explicitly, if we replace the mass matrices in (1.106) with the lumped CR mass matrix. Unfortunately, we have not been able to achieve satisfactory convergence results using the lumped matrix. We have, however, been able to speed up the Uzawa iteration significantly by using the "lumped" Schur complement as a preconditioner.

1.6.5 Numerical results

All numerical tests presented in this section were tests with analytic solution given by

$$\begin{aligned} u &= \sin(\pi x + \pi t)\sin(\pi y)\cos(\pi z), \\ v &= 2\cos(\pi t + \pi x)\cos(\pi y)\cos(\pi z), \\ w &= \cos(\pi t + \pi x)\sin(\pi y)\sin(\pi z), \\ p &= \frac{3\pi}{Re}\cos(\pi x + \pi t)\sin(\pi y)\cos(\pi z). \end{aligned} \tag{1.109}$$

The initial and the boundary conditions were given by the exact quantities and the source term was prescribed to satisfy the Navier-Stokes equations. The

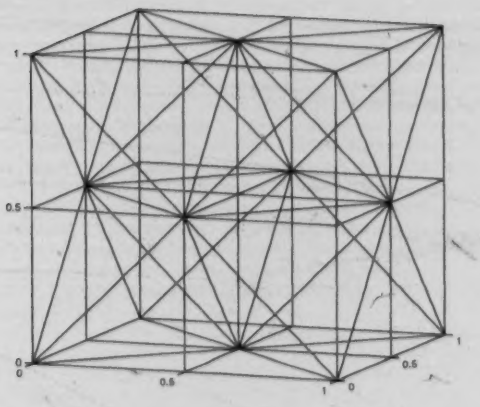

Figure 13: Sample $2\times2\times2$ grid.

Reynolds number was 100 in all tests. We used first order in time velocity-correction scheme.

The grids were generated by dividing the domain into hexagons and then dividing each hexagon into five tetrahedra using the diagonals of the six sides. There are two possible choices of diagonals to be used. We alternate them in each spatial direction in order to avoid effects similar to those on diagonal grids in two dimensions.

The domain for all tests was the cube $[0,1]\times[0,1]\times[0,1]$. We require that the number of divisions in each direction is even. This way we guarantee that each of the eight corners of the domain belongs to more than just one element. The simplest $2\times2\times2$ grid is illustrated in Figure 13.

Comparison of the implementations of the projection step. We performed one timestep on three different grids, solving the projection step in four different ways. In each case we recorded the dimension of the system, the number of iterations of the conjugate gradient method, and the processor time. The results are presented in Table 4. It is clear that by far the fastest method is the decoupled divergence-free basis, although it is the only one of the four that is not "exact." The only method that comes anywhere close is the system for the Lagrange multipliers solved with the preconditioner described in section 1.6.4. Using the preconditioner for the system of the full divergence-free basis (as explained in section 1.6.3) improves the times by approximately two thirds, although the latter method remains much slower than the preconditioned system for the Lagrange multipliers, which is our method of choice.

Table 4: Comparison of three implementations of the projection step in three dimensions. CPU times are given in seconds.

grid		8×8×8	16×16×16	32×32×32
FDF	dim	14680	111920	873568
	iters	1477	5236	16266
	CPU	11.88	381.99	9922.58
pFDF	dim	14680	111920	873568
	iters	36	37	40
	CPU	5.91	253.86	6208.91
pLM	dim	2559	20479	163839
	iters	9	10	10
	CPU	1.77	50.35	738.84
DDF	dim	2177	18945	157697
	iters	509	1550	4514
	CPU	0.36	13.44	355.96

FDF – full divergence-free basis
pFDF – full divergence-free basis with preconditioner
pLM – Lagrange multipliers with preconditioner
DFD – decoupled divergence-free basis

Table 5: Convergence of $\tilde{\mathbf{u}}_h$ for the exact and the decoupled projections.

		exact		decoupled	
Δt	grid	$\|\mathbf{u}-\tilde{\mathbf{u}}_h\|_0$	$\|\mathbf{u}-\tilde{\mathbf{u}}_h\|_1$	$\|\mathbf{u}-\tilde{\mathbf{u}}_h\|_0$	$\|\mathbf{u}-\tilde{\mathbf{u}}_h\|_1$
0.1	4×4×4	8.842624E−02	1.191992E+00	9.773791E−02	1.248917E+00
0.05	4×4×4	7.769892E−02	1.052004E+00	7.218177E−02	9.860881E−01
0.025	4×4×4	9.645550E−02	1.257096E+00	8.555169E−02	1.128029E+00
0.0125	4×4×4	1.144645E−01	1.494241E+00	1.024728E−01	1.335153E+00
0.00625	4×4×4	1.266868E−01	1.668811E+00	1.142835E−01	1.493773E+00
0.1	8×8×8	1.153971E−01	1.633659E+00	1.265216E−01	1.775108E+00
0.05	8×8×8	5.218383E−02	8.101185E−01	5.529173E−02	8.377576E−01
0.025	8×8×8	3.317029E−02	5.983091E−01	3.243121E−02	5.833688E−01
0.0125	8×8×8	3.236558E−02	6.309537E−01	3.066701E−02	5.966571E−01
0.00625	8×8×8	3.584356E−02	7.244892E−01	3.401954E−02	6.772170E−01
0.1	16×16×16	1.380253E−01	2.375558E+00	1.509637E−01	2.636312E+00
0.05	16×16×16	5.525217E−02	8.143109E−01	5.882347E−02	8.630148E−01
0.025	16×16×16	2.569275E−02	3.984662E−01	2.654278E−02	4.071030E−01
0.0125	16×16×16	1.350340E−02	2.625162E−01	1.358769E−02	2.629254E−01
0.00625	16×16×16	9.548250E−03	2.525905E−01	9.477206E−03	2.473674E−01
0.1	32×32×32	1.768688E−01	4.064747E+00	1.878966E−01	4.352278E+00
0.05	32×32×32	5.960215E−02	1.105334E+00	6.372980E−02	1.201064E+00
0.025	32×32×32	2.685252E−02	3.907454E−01	2.769251E−02	3.995507E−01
0.0125	32×32×32	1.304871E−02	1.985842E−01	1.318815E−02	2.003330E−01
0.00625	32×32×32	6.478678E−03	1.196336E−01	6.475063E−03	1.216127E−01

Convergence tests. We verified the implementation by conducting convergence tests. The test problem with an analytic solution was solved on a variety of grids and for a variety of timesteps. The computations were terminated at time $T = 1.0$. We used the same test to investigate the convergence and accuracy of the decoupled divergence-free projection and to compare it to the exact projection. The results are presented in Table 5. The norms are ℓ^2 in time. The computation of the exact projection was done using preconditioned Lagrange multipliers, since the solution it gives is identical to the one produced by the full solenoidal basis. We see that both projection methods give us $O(\Delta t + h^2)$ convergence in L^2-norm and $O(\Delta t + h)$ in H^1-norm. We also see that the accuracy of the decoupled projection is practically identical to the accuracy of the full system.

1.7 Conclusion

In the first part of the study, we devised a numerical method for solving the incompressible Navier-Stokes equations that is suitable for large scale simulations of free-surface flows. The method uses a conforming $\mathbb{P}_1$ element for the velocity to resolve the viscous step, and a nonconforming $\mathbb{P}_1$ interpolation for the velocity with $\mathbb{P}_0$ pressure at the projection step. The projected velocity is pointwise divergence-free in each element, thus it allows for improved mass conservation, which is important for the stability of the advection of free surfaces. The proposed numerical scheme does not produce an adequate approximation of the pressure; however, a fully convergent $\mathbb{P}_1$ pressure can be obtained at a postprocessing step via an appropriate Poisson problem.

We proved full order error estimate on grids, where the $\mathbb{P}_0$ pressure has a subspace that is inf-sup stable when paired with the conforming $\mathbb{P}_1$ velocity. Numerical results establish that the scheme is full order convergent even on "problematic" grids, where such pressure subspace fails to exist.

In two dimensions the projection step was implemented in three different ways and their performance was compared. Using Lagrange multipliers to impose incompressibility is the slowest. Relaxing the natural continuity of the velocity flux through element faces and imposing such continuity via additional inter-elemental Lagrange multipliers produces a 1.5 times larger system, which, however, takes slightly less time to invert. Our method of choice in two dimensions is the solenoidal approach, where a basis of the divergence-free subspace is constructed and used to implement the projection. This method gives the smallest system of equations, which also inverts much faster than the other two.

In three dimensions the solenoidal basis is fully coupled, unlike the two dimensional case, where the tangential components are orthogonal to the normal

components. This leads to a large linear system, which is very slow to invert. Decoupling the tangential and the normal components artificially introduces additional decoupling error, which, as shown by numerical results, is within the overall error of the method. The decoupled projection is by far the fastest projection method considered here, although it is unfit for solving multiphase problems with surface tension. Using the Lagrange multipliers in three dimensions requires a very slow Uzawa-type inner-outer iteration. If preconditioned with the solution of the same problem using the lumped CR mass matrix, the time taken by the Lagrange multipliers reduces significantly, which makes it our method of choice in three dimensions.

2 Discretization of problems with free capillary interfaces

2.1 Introduction

Numerical methods for direct computer simulations of multicomponent flows have been the subject of investigation for as long as Computational Fluid Dynamics has been around. Although the complexity of such flows has been limiting the range of attempted simulations to the simplest cases for a very long time, the recent increase in available computing power has induced an overwhelming amount of publications on this topic in the last decade or so.

Multiphase flows involve two or more immiscible fluids separated by a sharp interface. Although the flow of each fluid phase obeys the same physical law, they are, in general, described by different values of their physical parameters, such as density and viscosity, which leads to discontinuous coefficients of the governing PDEs. The possible smearing of the jumps in density and viscosity may lead to significant loss of accuracy. Things get even harder when surface tension is present, acting on the interfaces between fluid phases. Modeled as a Dirac δ-function, surface tension leads to irregularities in the exact solution. Moreover, surface tension depends on the interface's curvature, which introduces a strong nonlinearity in the equations and generally leads to problems with the stability of the numerical algorithms.

In the case of multicomponent flows, the governing equations of the flow are coupled with a scalar advection equation for the position of the free boundaries. The velocity of the system of fluids depends on the positions of the interfaces via the discontinuities of the coefficients, while the interfaces are advected with the fluid velocity. Even though there have been some attempts to solve the coupled system, the overwhelming majority of methods decouple the problem, solving the interface and the Navier-Stokes equations in separate substeps.

The numerical methods for simulations of flows involving moving boundaries can be divided loosely in two large classes. In the Eulerian frame of reference, the computations are performed on a fixed Cartesian grid, and some front-capturing method is utilized to describe the position of the interface. In the volume-of-fluid (VOF) method, introduced in [43], each phase is defined through an indicator function. A review of the VOF method can be found in [68]. We also refer the reader to [52, 53, 8] for more information about this approach. The advection of the step function inevitably produces computational cells that are partially filled with different fluids. In the volume-tracking

[0]A version of this chapter has been submitted for publication.
B. Bejanov, J.-L. Guermond, and P.D. Minev. A grid-alignment finite element technique for incompressible multicomponent flows. *J. Comput. Phys.*

method (see [66]), a sharp interface is reconstructed from the volume fractions in a way that allows computation of surface tension. See [25, 16, 65], among others, for recent developments in this technique. In [11], an application of the volume-tracking method is designed to handle situations where more than two fluids are present in the same computational cell.

The level-set method, introduced in [57] (see also [76, 74]), uses a continuous function, marking the interface with its zero level set. A commonly used marker is the distance function, since its gradient can be used to compute the curvature of the interface. Upon advecting it, however, the function still marks the interface with its zero level set, but its values at other points don't necessarily give the distance to the interface. For this reason, the distance function has to be reconstructed on every timestep. Reviews of the level-set method can be found in [56, 70, 71], and some recent implementations can also be found in [63, 60]. In [75, 73, 78], the level-set technique has been combined with the VOF method to overcome the difficulties level-set formulations usually have with mass conservation and to improve the overall accuracy.

An alternative to front-capturing are the front-tracking methods, first proposed in [89], where the interface is discretized with its own grid of co-dimension one. Here the interface grid moves with the flow, while the computational grid remains fixed. For recent developments and review of the front-tracking technique, see [87] and the numerous references therein. A grid-based front-tracking technique has been developed in [30, 31, 20], where the computational grid is modified locally to make a grid line follow the interface.

A common difficulty of Eulerian approaches lies in the treatment of the jumps in the coefficients and the δ-function of the surface tension. The interface generally intersects grid elements, and its position can be given at best within the resolution of the grid. As a result the interface is no longer sharp, but has finite thickness. The discontinuities get smeared and the Dirac δ-function gets smoothed over a few grid cells. This regularization, also known as continuous surface force (CSF, see [5]), is only first order accurate in space, and the use of high resolution grid or grid refinement near the interface becomes necessary. The immersed boundary method, proposed in [59], uses a set of discrete δ-functions to approximate the singular forces. The immersed interface method (see [50] and [51]) further develops the same idea and achieves a second order of accuracy.

The majority of Eulerian techniques are applied in Finite Difference or Finite Volume Method formulations. Some early examples of the use of the Finite Element Method can be found in [26, 54]. More recently, a method proposed in [55] utilizes a local enrichment of the finite element basis in the elements intersected by the interface in order to overcome the problems of CSF and improve mass conservation. A different approach is presented in [9, 10],

where an adaptive Eulerian grid is used, in the sense that on each timestep the grid is temporarily aligned with the interface via local grid refinement. This way the interface is kept sharp, and the interpolation is optimal.

Another possibility for the choice of frame of reference is the Lagrangian approach, where grid nodes are associated with fluid particles and move with the flow. This approach is very useful in simulations of plasticity, where flows are slow and deformations are small. When applied to viscous flows, however, this method usually experiences difficulties, since the grid easily becomes heavily distorted and inadequate for computations. These situations require remeshing, which is a rather expensive procedure that adds significantly to the computational cost of the algorithm. To address this issue, the arbitrary Lagrangian-Eulerian (ALE) technique was developed (see [42]). The general idea is to use some other velocity to advect the mesh nodes. The purely Lagrangian and the Eulerian approaches can be seen as particular cases of ALE, where the grid velocity is zero in the Eulerian formulation and equal to the fluid velocity in the purely Lagrangian one. An overview of the ALE method can be found in [21] and the references therein. Examples of recent development in this class of methods are [58, 18, 62]. The deforming-spatial-domain/stabilized space-time (DSD/SST) method developed in [83, 84] is an interface-tracking technique, where the finite element formulation of the problem is written over its space-time domain and the equation of elasticity is solved to move the mesh. Even though this and various other stabilization techniques can reduce the frequency of the needed remeshings in many cases, there are still many situations where remeshing is needed too often to be feasible. Recent developments in [81, 82] present a method featuring a fusion of the front-tracking, applied in ALE setting, and the Eulerian front-capturing techniques. It uses front-tracking when remeshing is not needed too often, and front-capturing otherwise.

The method developed in the current study best fits in the class of ALE. It is an attempt to eliminate the need for costly remeshing completely. The idea has evolved from the notion of adaptive Eulerian grids (see [9, 10]). Instead of using local refinement, which introduces a different set of additional degrees of freedom on each timestep, we have developed a technique, which we will call local grid alignment. We maintain a reference grid, which is used on each timestep to generate a new temporary grid that is aligned with the interface. To do this, we project a small number of nodes that are close to the current position of the interface onto it. The resulting computational grid has the same connectivity as the reference grid; this allows for an easier and more effective parallelization. Moreover, in simulations of the motion of a large number of interfaces, the alignment is an intrinsically parallel procedure, since different interfaces do not intersect each other and each interface can be treated

separately.

Since the computational grid is aligned with the interface on every timestep, piecewise linear approximation of the current position of the interface is embedded into the current computational grid as a set of nodes and faces. Adding to the benefit of optimal interpolation is the possibility to impose the boundary condition on the interface in the usual for finite elements way. To compute the surface tension, we use a simple and cost-effective procedure based on a local approximation of the surface with a circle (in two dimensions) or sphere (in three dimensions). This approach provides an approximation that is smooth enough to compute accurately the mean curvature, while avoiding the need to solve a linear system, which is done when a smooth curve is interpolated or fitted globally (a cubic spline, for example). The same local circles/spheres are also used to calculate the projections of the nodes during the alignment procedure.

This chapter is organized as follows. In section 2.2 we formulate the multiphase problem and introduce notation. Section 2.3 presents space and time discretizations and outlines a time-marching algorithm. The following two sections 2.4 and 2.5 are dedicated to the grid alignment itself and various other issues of implementation in two and three dimensions correspondingly. These sections also present numerical validations for the proposed algorithms. The chapter concludes with a brief summary in section 2.6.

2.2 Formulation of the problem and notation

We consider an open, bounded, and connected domain Ω, whose boundary is smooth enough for our purposes. Since we are interested in the simulations of bubbles (to be concise, we say bubble to mean bubble or droplet), we refer to the background phase as phase 0 and assign to the bubbles consecutive indices starting from 1. For simplicity, we discuss the case of one bubble.

To distinguish between fluids, we use a lower index $o \in \{0, 1\}$. Thus, Ω_o denotes the open region occupied by fluid o. We assume that $\Omega_0 \cap \Omega_1 = \emptyset$ and $\overline{\Omega} = \overline{\Omega}_0 \cup \overline{\Omega}_1$. We also assume that Ω_0 is connected, Ω_1 is simply connected, and denote by $\Gamma_1 = \partial\Omega_0 \cap \partial\Omega_1$ the interface between bubble 1 and the background phase 0. We assume that $\Gamma_1 \cap \partial\Omega = \emptyset$ at all times. Since we consider only one bubble, we drop the index and refer to Γ as "the interface."

The fluids are considered to be immiscible, viscous, Newtonian; therefore, they obey the incompressible Navier-Stokes equations.

$$\rho_o \frac{\partial \mathbf{u}_o}{\partial t} + \rho_o \left(\mathbf{u}_o \cdot \nabla\right) \mathbf{u}_o - \mu_o \nabla^2 \mathbf{u}_o + \nabla p_o = \rho_o \mathbf{f}_o \qquad \text{in } \Omega_o, \tag{2.1}$$

$$\nabla \cdot (\rho_o \mathbf{u}_o) = 0 \qquad \text{in } \Omega_o. \tag{2.2}$$

The balance of forces on the interface is expressed with the following equation:

$$\left(p_1\mathbf{I} - p_0\mathbf{I} - \mu_1\nabla\mathbf{u}_1 + \mu_0\nabla\mathbf{u}_0\right)\cdot\mathbf{n} = \sigma\varkappa\mathbf{n} \qquad \text{on } \Gamma. \tag{2.3}$$

We also assume that the velocity is subject to initial and boundary conditions as in (1.4),(1.3). In the equations above $\rho_o, \mu_o, \mathbf{u}_o, p_o$, and $\mathbf{f}_o$ are the density, viscosity, velocity, pressure, and external force of fluid o. From now on, when the index o is missing, we mean the quantity defined piecewise, e.g.

$$\mathbf{u} = \begin{cases} \mathbf{u}_0 & \text{in } \Omega_0, \\ \mathbf{u}_1 & \text{in } \Omega_1. \end{cases}$$

In (2.3) $\mathbf{n}$ is the unit vector, which is normal to the interface and pointing out of Ω_1, $\varkappa$ is the mean curvature of Γ, σ is the coefficient of surface tension, and $\mathbf{I}$ is the identity tensor.

Nondimensionalization. We choose some appropriate characteristic time interval t_c, length l_c, speed u_c, and pressure p_c, and nondimensionalize the equations using

$$t = t_c\bar{t}, \quad \mathbf{x} = l_c\bar{\mathbf{x}}, \quad \mathbf{u} = u_c\bar{\mathbf{u}}, \quad p = p_c\bar{p}. \tag{2.4}$$

The momentum equation in nondimensional variables becomes

$$\frac{\rho u_c}{t_c}\frac{\partial\bar{\mathbf{u}}}{\partial\bar{t}} + \frac{\rho u_c^2}{l_c}\left(\bar{\mathbf{u}}\cdot\bar{\nabla}\right)\bar{\mathbf{u}} - \frac{\mu u_c}{l_c^2}\bar{\nabla}^2\bar{\mathbf{u}} + \frac{p_c}{l_c}\bar{\nabla}\bar{p} = \rho\hat{\mathbf{f}} \quad \text{in } \Omega_0\cup\Omega_1, \tag{2.5}$$

where $\hat{\mathbf{f}}(\bar{\mathbf{x}}) = \mathbf{f}(l_c\bar{\mathbf{x}})$ and $\bar{\nabla}$ denotes differentiation with respect to the nondimensional spatial variables $\bar{\mathbf{x}}$. Now we divide (2.5) by $\dfrac{\rho_0 u_c^2}{l_c}$ and set $l_c = t_c u_c$ and $p_c = \rho_0 u_c^2$. The resulting equation is

$$\frac{\rho}{\rho_0}\frac{\partial\bar{\mathbf{u}}}{\partial\bar{t}} + \frac{\rho}{\rho_0}\left(\bar{\mathbf{u}}\cdot\bar{\nabla}\right)\bar{\mathbf{u}} - \frac{\mu}{\mu_0}\frac{\mu_0}{\rho_0 u_c l_c}\bar{\nabla}^2\bar{\mathbf{u}} + \bar{\nabla}\bar{p} = \frac{\rho}{\rho_0}\frac{l_c}{u_c^2}\hat{\mathbf{f}} \quad \text{in } \Omega_0\cup\Omega_1. \tag{2.6}$$

The Reynolds number is taken from the background phase and is given by

$$Re = \frac{\rho_0 u_c l_c}{\mu_0}. \tag{2.7}$$

We set the nondimensional external force to $\bar{\mathbf{f}} = \frac{l_c}{u_c^2}\hat{\mathbf{f}}$. We use λ_ρ and λ_μ to denote the ratios of density and viscosity of the fluids, i.e.

$$\lambda_\rho = \frac{\rho}{\rho_0} = \begin{cases} 1 & \text{in } \Omega_0, \\ \frac{\rho_1}{\rho_0} & \text{in } \Omega_1, \end{cases} \qquad \lambda_\mu = \frac{\mu}{\mu_0} = \begin{cases} 1 & \text{in } \Omega_0, \\ \frac{\mu_1}{\mu_0} & \text{in } \Omega_1. \end{cases} \tag{2.8}$$

Substituting all these into (2.6), we obtain the nondimensionalized momentum equation

$$\lambda_\rho \frac{\partial \bar{\mathbf{u}}}{\partial \bar{t}} + \lambda_\rho \left(\bar{\mathbf{u}} \cdot \bar{\nabla}\right) \bar{\mathbf{u}} - \lambda_\mu \frac{1}{Re} \bar{\nabla}^2 \bar{\mathbf{u}} + \bar{\nabla} \bar{p} = \lambda_\rho \bar{\mathbf{f}} \qquad \text{in } \Omega_0 \cup \Omega_1. \tag{2.9}$$

Before we continue, suppose the external force is due to gravity, i.e.

$$\mathbf{f} = g\mathbf{e}_g, \tag{2.10}$$

where g is the gravitational acceleration and $\mathbf{e}_g$ is the unit vector in direction of gravity. Then the nondimensional gravity is given by

$$\bar{\mathbf{f}} = \frac{1}{Fr}\mathbf{e}_g, \tag{2.11}$$

where the Froude number is defined as

$$Fr = \frac{u_c^2}{l_c g}. \tag{2.12}$$

Next, we nondimensionalize the interface condition. We start by substituting (2.4) into (2.3). Taking into account that nondimensionalizing the curvature gives $\varkappa = \frac{1}{l_c}\bar{\varkappa}$, and using the relations above, we arrive at

$$\left[(\bar{p}_1 - \bar{p}_0)\mathbf{I} - \frac{1}{Re}\left(\frac{\mu_1}{\mu_0}\bar{\nabla}\bar{\mathbf{u}}_1 - \bar{\nabla}\bar{\mathbf{u}}_0\right)\right] \cdot \mathbf{n} = \frac{\sigma}{\rho_0 u_c^2 l_c}\bar{\varkappa}\mathbf{n} \quad \text{on } \Gamma. \tag{2.13}$$

The Weber number is defined as

$$We = \frac{\rho_0 u_c^2 l_c}{\sigma}. \tag{2.14}$$

Thus the nondimensional force balance on the interface becomes

$$\left[(\bar{p}_1 - \bar{p}_0)\mathbf{I} - \frac{1}{Re}\left(\frac{\mu_1}{\mu_0}\bar{\nabla}\bar{\mathbf{u}}_1 - \bar{\nabla}\bar{\mathbf{u}}_0\right)\right] \cdot \mathbf{n} = \frac{1}{We}\bar{\varkappa}\mathbf{n} \quad \text{on } \Gamma. \tag{2.15}$$

For simpler notation, from now on we drop the bars with the understanding that all quantities and equations are properly nondimensionalized.

Weak formulation. Let us multiply equation (2.9) by a test function $\mathbf{v}$ with compact support and integrate over Ω. The integration by parts of the diffusion and the pressure terms is carried out as follows:

$$
\begin{aligned}
&-\lambda_\mu \frac{1}{Re}\int_\Omega \nabla^2\mathbf{u}\cdot\mathbf{v}\,dx + \int_\Omega \nabla p\cdot\mathbf{v}\,dx \\
&\quad = -\frac{1}{Re}\int_{\Omega_0}\nabla^2\mathbf{u}_0\cdot\mathbf{v}\,dx - \frac{\mu_1}{\mu_0}\frac{1}{Re}\int_{\Omega_1}\nabla^2\mathbf{u}_1\cdot\mathbf{v}\,dx \\
&\qquad\quad + \int_{\Omega_0}\nabla p_0\cdot\mathbf{v}\,dx + \int_{\Omega_1}\nabla p_1\cdot\mathbf{v}\,dx \\
&\quad = \frac{1}{Re}\int_{\Omega_0}\nabla\mathbf{u}_0 : \nabla\mathbf{v}\,dx + \frac{1}{Re}\int_\Gamma(\nabla\mathbf{u}_0\cdot\mathbf{v})\cdot\mathbf{n}\,ds \\
&\qquad\quad + \frac{\mu_1}{\mu_0}\frac{1}{Re}\int_{\Omega_1}\nabla\mathbf{u}_1 : \nabla\mathbf{v}\,dx - \frac{\mu_1}{\mu_0}\frac{1}{Re}\int_\Gamma(\nabla\mathbf{u}_1\cdot\mathbf{v})\cdot\mathbf{n}\,ds \\
&\qquad\quad - \int_{\Omega_0} p_0\nabla\cdot\mathbf{v}\,dx - \int_\Gamma p_0\mathbf{v}\cdot\mathbf{n}\,ds - \int_{\Omega_1} p_1\nabla\cdot\mathbf{v}\,dx + \int_\Gamma p_1\mathbf{v}\cdot\mathbf{n}\,ds \\
&\quad = \lambda_\mu\frac{1}{Re}\int_\Omega\nabla\mathbf{u}:\nabla\mathbf{v}\,dx - \int_\Omega p\nabla\cdot\mathbf{v}\,dx \\
&\qquad\quad + \int_\Gamma\left\{\left[p_1\mathbf{I} - p_0\mathbf{I} - \frac{1}{Re}\left(\frac{\mu_1}{\mu_0}\nabla\mathbf{u}_1 - \nabla\mathbf{u}_0\right)\right]\cdot\mathbf{n}\right\}\cdot\mathbf{v}\,ds \\
&\quad = \lambda_\mu\frac{1}{Re}\int_\Omega\nabla\mathbf{u}:\nabla\mathbf{v}\,dx - \int_\Omega p\nabla\cdot\mathbf{v}\,dx + \frac{1}{We}\int_\Gamma \varkappa\mathbf{n}\cdot\mathbf{v}\,ds.
\end{aligned}
$$

This calculation can be used to justify the following weak formulation of the problem: *find* $\mathbf{u}\in\mathbf{H}^1$ *satisfying the initial and the boundary conditions* (1.4) *and* (1.3), *and pressure* $p\in L^2/\mathbb{R}$, *such that for all* $\mathbf{v}\in\mathbf{H}^1_0$ *and* $q\in L^2$,

$$
\lambda_\rho\left(\frac{\partial\mathbf{u}}{\partial t},\mathbf{v}\right) + \lambda_\rho\left((\mathbf{u}\cdot\nabla)\,\mathbf{u},\mathbf{v}\right) + \frac{\lambda_\mu}{Re}\left(\nabla\mathbf{u},\nabla\mathbf{v}\right) - \left(p,\nabla\cdot\mathbf{v}\right) \qquad (2.16)
$$

$$
= \lambda_\rho\left(\mathbf{f},\mathbf{v}\right) - \frac{1}{We}\int_\Gamma \varkappa\mathbf{v}\cdot\mathbf{n}\,ds,
$$

$$
\left(\nabla\cdot\mathbf{u},q\right) = 0. \qquad (2.17)
$$

2.3 Discretization

We use the same spatial and temporal discretization as discussed in sections 1.3.1 and 1.3.2. Here we focus only on the specifics of the treatment

of the moving interface.

2.3.1 Spatial discretization

For each time level $n \in \{0, \ldots, \frac{T}{\Delta t}\}$, we have a grid $\mathcal{G}_{h,n} = (\mathcal{N}_{h,n}, \mathcal{F}_{h,n}, \mathcal{T}_{h,n})$, on which all computations are performed. We use a subscript n to denote the time level at which the given grid is used. Also, in the sequel, for a given time-dependent discrete quantity r_h, we use a superscript m to denote the time level at which the quantity is taken, and a subscript n to denote the grid on which the quantity is based, i.e. $r_{h,n}^m$ denotes r_h at time $t_m = m\Delta t$ on grid $\mathcal{G}_{h,n}$.

The purpose of developing a method that features low-order velocity and discontinuous pressure is to model accurately the jumps in the spatial derivatives of the velocity and the pressure across the moving interface. In order to take advantage of this, as well as the fact that we have a locally mass-conserving approximation of the velocity, we need the computational grid to be aligned with the boundary between different phases. Since this boundary moves with time, the grid will also have to change from one timestep to the next to reflect the new position of the interface. Also, we require that the change in computational grid is only in the positions of the nodes, but no change is done in the connectivity. In other words, we allow neither the introduction nor the removal of any nodes, faces, elements, or in general degrees of freedom. Although we will have to reassemble all matrices on each timestep, the structure of the matrices will remain unchanged, which is desirable to make parallelization easier and more efficient. In addition, if we confine the alignment of the grid to moving only a small number of nodes, then only a small number of entries in these matrices will need to be updated.

We maintain a reference grid $\mathcal{R}_h = (\mathcal{N}_h, \mathcal{F}_h, \mathcal{T}_h)$ and use it to produce the computational grids $\mathcal{G}_{h,n} = (\mathcal{N}_{h,n}, \mathcal{F}_{h,n}, \mathcal{T}_{h,n})$. If we know the current position of the interface, we can find the nodes in $\mathcal{N}_h$ that are closest to it and move some of them only a small distance ($< h$) to position them on Γ. The grid $\mathcal{G}_{h,n}$, produced this way, is aligned with the interface, in the sense that Γ does not intersect any faces (in two dimensions) or edges (in three dimensions). The faces in $\mathcal{F}_{h,n}$ (both in two and three dimensions) that have all their nodes on the interface form a piecewise linear interpolation of Γ. We denote this interpolant by $\Gamma_{h,n}^n$. Notice that $\mathcal{G}_{h,n}$ has the same connectivity as $\mathcal{R}_h$, therefore all grids have the same connectivity.

2.3.2 Temporal discretization

As before, we consider a finite time interval $[0, T]$, which we divide into timesteps of equal length Δt via time levels $t_n = n\Delta t$ for $n = 0, 1, \ldots, \frac{T}{\Delta t}$. The

proposed time-marching algorithm is based on the velocity-correction scheme developed in the first chapter.

Suppose that at time $t = 0$ we have a grid $\mathcal{G}_{h,0}$, which is aligned with the initial position of the interface. $\Gamma^0_{h,0}$ is the piecewise linear interpolant of Γ embedded into $\mathcal{G}_{h,0}$. We also assume that we have velocities $\tilde{\mathbf{u}}^0_{h,0} \in \mathbf{X}_{h,0}$ and $\mathbf{u}^0_{h,0} \in \mathbf{V}_{h,0}$, which are appropriate approximations of the initial condition for the velocity in their corresponding spaces. For each $n = 0, \ldots, \left(\frac{T}{\Delta t} - 1\right)$, we perform the following steps to advance from $t = t_n$ to $t = t_{n+1}$.

1. **Calculation of the new position of the interface.** We advect the positions of all nodes in $\mathcal{G}_{h,n}$ with the divergence-free velocity $\mathbf{u}^n_{h,n}$. Let $\mathbf{x}^n_{h,n}$ be the *identity* element in $\mathbf{X}_{h,n}$, in the sense that it takes at each node in $\mathcal{N}_{h,n}$ the value equal to the coordinates of this node at time t_n. Then we *find* $\mathbf{x}^{n+1}_{h,n} \in \mathbf{X}_{h,n}$ *such that for all* $\mathbf{v}_h \in \mathbf{X}_{h,n}$, *we have*

$$\frac{1}{\Delta t}\left(\mathbf{x}^{n+1}_{h,n} - \mathbf{x}^n_{h,n}, \mathbf{v}\right) - \left(\left(\mathbf{u}^n_{h,n}\cdot\nabla\right)\mathbf{x}^n_{h,n}, \mathbf{v}\right) = 0. \tag{2.18}$$

   We will refer to the new node positions $\mathbf{x}^{n+1}_{h,n}$ as the *advected* $\mathcal{G}_{h,n}$. The new positions of the interface nodes give us an approximation $\Gamma^{n+1}_{h,n}$ of the position of the interface at time t_{n+1}. The advected $\mathcal{G}_{h,n}$ is not used for computations, but only to define $\Gamma^{n+1}_{h,n}$ for the purpose of grid alignment.

2. **Grid alignment.** Using the reference grid $\mathcal{R}_h$ and the new position of the interface $\Gamma^{n+1}_{h,n}$, we generate a new grid $\mathcal{G}_{h,n+1}$. The approximation of the interface embedded into $\mathcal{G}_{h,n+1}$ is, in general, different from $\Gamma^{n+1}_{h,n}$, and we denote it $\Gamma^{n+1}_{h,n+1}$. The details of this procedure are presented in section 2.4 for the two-dimensional case and section 2.5 for the three-dimensional case.

3. **Transfer of data to the new grid.** $\tilde{\mathbf{u}}^n_{h,n}$ is an approximation of $\mathbf{u}^n$ based on the old grid $\mathcal{G}_{h,n}$. In this step we compute $\tilde{\mathbf{u}}^n_{h,n+1}$, which is an appropriate approximation of the same quantity $\mathbf{u}^n$, but based on the new grid $\mathcal{G}_{h,n+1}$. Further discussion of this step follows shortly (section 2.3.3).

4. **Projection step.** *Find* $\mathbf{u}^{n+1}_{h,n+1} \in \mathbf{V}_{h,n+1}$ *such that for all* $\mathbf{v}_h \in \mathbf{V}_{h,n+1}$,

$$\begin{aligned}\frac{\lambda_\rho}{\Delta t}\left(\mathbf{u}^{n+1}_{h,n+1} - \tilde{\mathbf{u}}^n_{h,n+1}, \mathbf{v}_h\right) = \lambda_\rho\left(\mathbf{f}^{n+1}, \mathbf{v}_h\right) - \lambda_\rho\left(\left(\tilde{\mathbf{u}}^n_{h,n+1}\cdot\nabla\right)\tilde{\mathbf{u}}^n_{h,n+1}, \mathbf{v}_h\right) \\ - \frac{\lambda_\mu}{Re}\left(\nabla\tilde{\mathbf{u}}^n_{h,n+1}, \nabla\mathbf{v}_h\right) - \frac{1}{We}\int_\Gamma \varkappa\mathbf{v}_h\cdot\mathbf{n}\, ds.\end{aligned} \tag{2.19}$$

5. **Velocity-correction step.** *Find* $\tilde{\mathbf{u}}_{h,n+1}^{n+1} \in \mathbf{X}_{h,n+1}$ *such that for all* $\tilde{\mathbf{v}}_h \in \mathbf{X}_{h,n+1}$, *we have:*

$$\frac{\lambda_\rho}{\Delta t}\left(\tilde{\mathbf{u}}_{h,n+1}^{n+1} - \mathbf{u}_{h,n+1}^{n+1}, \tilde{\mathbf{v}}_h\right) + \frac{\lambda_\mu}{Re}\left(\nabla\left(\tilde{\mathbf{u}}_{h,n+1}^{n+1} - \tilde{\mathbf{u}}_{h,n+1}^{n}\right), \nabla\tilde{\mathbf{v}}_h\right) = 0. \quad (2.20)$$

2.3.3 Transfer of data between grids

The simplest option is to employ the usual finite element interpolation, and obtain the nodal values of $\tilde{\mathbf{u}}_{h,n+1}^{n}$ by interpolating the values of $\tilde{\mathbf{u}}_{h,n}^{n}$ at the nodes of $\mathcal{G}_{h,n+1}$. This procedure is straightforward; obviously, it produces an approximation within the error of the method.

A concern with this approach arises when there is a large ratio of viscosities. In this case, the velocity exhibits a jump discontinuity in its spatial derivatives, and this may result in a loss of an order of accuracy of interpolation at nodes near or on the interface. However, if the timestep is sufficiently small, there will never be any nodes changing phases between successive grids. A node that is near the interface in $\mathcal{G}_{h,n}$ will either remain in the same phase in $\mathcal{G}_{h,n+1}$ or move on $\Gamma_{h,n+1}^{n+1}$, but will not "jump" over the interface into the other phase. This way any reinterpolation takes place within the same fluid phase, where the exact velocity is smooth, and full order of accuracy is achieved. We do anticipate, however, that the error of interpolation will be greater for larger viscosity ratios.

When interpolation is used to transfer data between grids, the interpolated velocity $\tilde{\mathbf{u}}_{h,n+1}^{n}$ is computed and can be used directly for (2.19) and (2.20). In this case, we practically solve an Eulerian formulation on each timestep, although on a different grid every time.

An alternative to direct interpolation is the following Lagrangian approach, which is the standard in ALE formulations (see, for example, [19]). All grids $\mathcal{G}_{h,n}$ have the same connectivity as the reference grid $\mathcal{R}_h$, from which they were produced; therefore, there is a natural one-to-one correspondence between the nodes in different grids. We identify the nodes by their indices. Thus if $\mathbf{x}_i \in \mathcal{N}_h$ is a node in the reference grid, then we denote by $\mathbf{x}_i^n \in \mathcal{N}_{h,n}$ the position of the same node at time $t = t_n$, i.e. in grid $\mathcal{G}_{h,n}$. Now we define the discrete mesh velocity via a forward difference. For each node $\mathbf{x}_i$, we have

$$\hat{\mathbf{u}}_i^n = \frac{\mathbf{x}_i^{n+1} - \mathbf{x}_i^n}{\Delta t}. \quad (2.21)$$

This discrete velocity can be extended to a piecewise linear velocity using the standard finite element interpolation. For this purpose, we define $\hat{\mathbf{u}}_{h,n+1}^{n} \in$

$\mathbf{X}_{h,n+1}$ such that

$$\hat{\mathbf{u}}^n_{h,n+1}(\mathbf{x}^{n+1}_i) = \hat{\mathbf{u}}^n_i \qquad \forall \mathbf{x}^{n+1}_i \in \mathcal{N}_{h,n+1}. \tag{2.22}$$

We also define the transferred material velocity $\tilde{\mathbf{u}}^n_{h,n+1} \in \mathbf{X}_{h,n+1}$ as

$$\tilde{\mathbf{u}}^n_{h,n+1}(\mathbf{x}^{n+1}_i) = \tilde{\mathbf{u}}^n_{h,n}(\mathbf{x}^n_i). \tag{2.23}$$

With these definitions, we are ready to write down the ALE formulation. The projection step (2.19) becomes: *find* $\mathbf{u}^{n+1}_{h,n+1} \in \mathbf{V}_{h,n+1}$ *such that for all* $\mathbf{v}_h \in \mathbf{V}_{h,n+1}$,

$$\begin{aligned}\frac{\lambda_\rho}{\Delta t}\left(\mathbf{u}^{n+1}_{h,n+1} - \tilde{\mathbf{u}}^n_{h,n+1}, \mathbf{v}_h\right) &= \lambda_\rho\left(\mathbf{f}^{n+1}, \mathbf{v}_h\right) \\ &\quad - \lambda_\rho\left(\left(\tilde{\mathbf{u}}^n_{h,n+1} - \hat{\mathbf{u}}^n_{h,n+1}\cdot\nabla\right)\tilde{\mathbf{u}}^n_{h,n+1}, \mathbf{v}_h\right) \\ &\quad - \frac{\lambda_\mu}{Re}\left(\nabla\tilde{\mathbf{u}}^n_{h,n+1}, \nabla\mathbf{v}_h\right) - \frac{1}{We}\int_\Gamma \varkappa\mathbf{v}_h\cdot\mathbf{n}\, ds,\end{aligned} \tag{2.24}$$

while the velocity-correction step (2.20) remains unchanged.

2.4 Grid alignment in two spatial dimensions

In this section $N = 2$. Here we will use the terms *edge* and *face* interchangeably, since in two dimensions the edges connecting grid nodes and the faces of the elements are the same objects (i.e. $\mathcal{E}_h = \mathcal{F}_h$). However, our use of these terms in this section will be consistent with their use in the description of the algorithm in three dimensions (where $\mathcal{E}_h \neq \mathcal{F}_h$). Also, for simplicity of notation, we will omit the subscript n denoting the time level when, in our opinion, there is no risk of ambiguity or confusion.

2.4.1 Alignment algorithm

We have the reference grid $\mathcal{R}_h$ and a piecewise linear approximation Γ_h of the interface Γ. The task is to produce a new grid $\mathcal{G}_h$, which has the same connectivity as $\mathcal{R}_h$ and does not contain any edges intersected by the interface. The proposed algorithm proceeds as follows.

1. Create lists of intersected edges and pending nodes. We determine the phase to which each node in $\mathcal{R}_h$ belongs. Edges having end nodes in different phases are intersected by Γ. We denote the set containing all intersected edges by $\mathcal{X}_h \subset \mathcal{E}_h$. We also denote by $\mathcal{P}_h \subset \mathcal{N}_h$ the set of all nodes that are endpoints of edges in $\mathcal{X}_h$. We refer to the nodes in $\mathcal{P}_h$ as pending nodes,

since some of them will be moved on the interface, while others will remain in their places. In addition, we maintain an edge counter associated with each pending node. The value of this counter is (and will be at all times until the end of step 3) equal to the number of edges in $\mathcal{X}_h$ containing the pending node.

Note that no node is marked as an interface node at this time. If the reference position of some node happens to be on the interface, we mark this node as belonging to one of the phases. Such node will necessarily end up in $\mathcal{P}_h$, since it has adjacent nodes on both sides of the interface. If this were not the case, then there would be a face intersected at least twice by Γ, which would mean that the grid resolution is not fine enough to resolve such an interface.

2. Sort the pending nodes list. For each node $P \in \mathcal{P}_h$, we compute its projection onto the interface, i.e. we find a point $P' \in \Gamma$ with the shortest distance to P. Of course, the distance between P and P' is the distance from P to Γ. We sort the pending nodes by their distance to the interface, with the node closest to the interface being first and the farthest being last in out list.

3. Move nodes onto the interface. This step is where the actual alignment takes place. We process nodes in $\mathcal{P}_h$ one at a time in the order in which they were put in step 2. Let $P \in \mathcal{P}_h$ be the node we are currently handling.

- If the edge counter of P is zero, then we remove P from $\mathcal{P}_h$ and move on to the next node.
- If P is on $\partial\Omega$, then we cannot move it, so we do nothing and move on to the next node.
- Otherwise, we move P to the position of its projection P' on Γ. The node is then removed from $\mathcal{P}_h$, and all intersected edges containing P are removed from $\mathcal{X}_h$, since they are no longer intersected (see Figure 14). When we remove an edge from $\mathcal{X}_h$, we make sure to decrease by one the edge counters of its two nodes.

At the end of this procedure, there are no intersected edges left, i.e. $\mathcal{X}_h$ is now empty. A by-product is that any nodes that may still be in $\mathcal{P}_h$ have their edge counters equal to zero and are not pending anymore. We now have a "first draft" of $\mathcal{G}_h$.

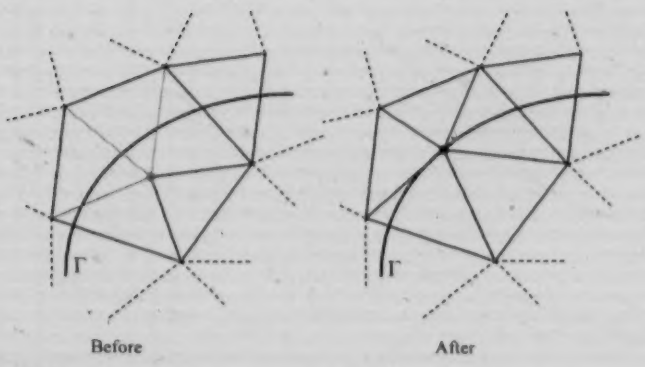

Figure 14: Projecting a pending node onto the interfaces. In the new position, no edge containing the node intersects Γ.

4. Nodes consistency. This is the first of a series of verifications of the consistency and the quality of the grid. With each node in $\mathcal{G}_h$ we associate the polygon with vertices at the nodes adjacent to it. If the node belongs to its polygon, then all elements containing this node have positive Jacobians. Otherwise, we move the node to the centroid of its polygon and thus untangle the part of the grid connected to it. We need to perform this check only for nodes that were projected onto the interface and their immediate neighbors, since only elements containing these nodes were changed during step 3.

If the node we need to untangle is on the interface, then, of course, it will no longer be on Γ after moving it to a new position. This may reintroduce intersected edges, which we do not allow. In order to avoid reintroducing intersected edges, a node is removed from the interface only if all nodes adjacent to it, which are not on Γ, belong to the same phase. Then, in its new position at the centroid of its polygon, the node will belong to the phase of its neighbors and neither one of its edges will be intersected. Otherwise, the node is left tangled for now. Since some adjacent nodes must necessarily be outside their polygons, untangling some of them will fix the polygon of this node as well.

5. Elements consistency. We do not allow elements to have three nodes on the interface, since it is impossible to decide to which phase such element belongs. For each element incident with Γ, we count the number of interface nodes in the element. If this number is 3, then we remove one of the element's nodes from the interface and place it in the centroid of the node's polygon. To choose a node to be removed from the interface, we follow the same rule as in step 4. At least one of the element's nodes will have all of its neighbors on the same side of the interface, except for the two other element

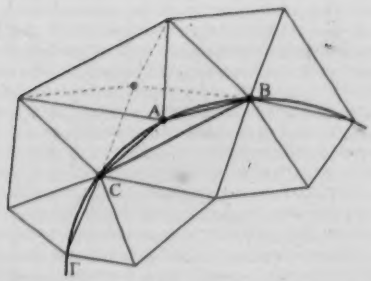

Figure 15: Element with three nodes on Γ. At least one node (A) will have all neighbors on the same side of Γ.

nodes, which are on Γ (see Figure 15).

6. Interface consistency. We verify the integrity of the approximation of the interface embedded into the new grid by counting the number of adjacent interface nodes for each interface node. This number must be exactly 2 for all nodes when the interface is closed. Any node with 0 or 1 neighbors on the interface is removed from it and placed in the centroid of its polygon. If a node is connected to more than 2 interface nodes, then the algorithm fails. This means that the interface intersects itself at this point. Such situation occurs, for example, when pinching starts to develop. Since it is not our goal to model this phenomenon, if this happens, the current algorithm simply stops. If needed, the handling of topology change in the interface should be placed at this point in the algorithm.

The state of $\mathcal{G}_h$ at the end of this step is suitable for computations. We add one more step, which our numerical experience has demonstrated to improve the stability and the accuracy of the method.

7. Volume adjustment. The piecewise linear interpolation of the interface embedded in $\mathcal{G}_h$ is not the same as the one with which we started. This is due to the fact that the interface nodes in the new grid do not necessarily coincide with the vertices of Γ_h. Thus, in general, the volume of the bubble in $\mathcal{G}_h$ is slightly different. We compensate for this mass loss or gain by moving all interface nodes away from or towards the center of mass of the interior phase, until the proper volume is restored.

After the correction of the volume, some elements near the boundary may become intolerably stretched, or even have negative Jacobians. We correct this by another pass over the nodes near the interface, placing them in the centroids of their polygons.

2.4.2 Some remarks on the alignment algorithm

Moving a node to the centroid of its polygon is the simplest technique for improvement of the quality of a mesh; it is known in the literature as Laplacian smoothing. Intuitively, it is clear that there is a problem if the polygon of the node is not convex and its centroid is actually outside. Indeed, it is well known that Laplacian smoothing improves the quality of the grid only slightly and can (and often does) decrease the quality of the elements, and can even render the mesh invalid. A simple remedy is to use what is called "smart" Laplacian smoothing, where the node is moved to the centroid only if this would actually improve the minimum quality of the surrounding elements. This ensures that the quality of the grid will not decrease. More substantial improvement of the quality of the grid can be achieved by optimization-based local mesh smoothing, where the node is placed at the position within its polygon that gives the maximum possible minimal quality of the surrounding elements. The solution of this optimization problem for each node that needs to be adjusted, obviously, causes these methods to be significantly more computationally intensive than the simple Laplacian smoothing. In all tests conducted so far, we have not encountered a situation that the Laplacian smoothing has not been able to handle. For more information on local mesh smoothing techniques the reader is referred to [28, 27].

To measure the quality of the elements, we use the ratio of the radii of the incircle and the circumcircle of the triangle. For a comprehensive review and comparison of various element quality measures we refer to [24] and the references therein.

The first three steps of the algorithm require solutions for two problems – determining the phase to which given point belongs, and computing the projection of a given point on the interface. At time $t = 0$, when producing the initial grid $\mathcal{G}_{h,0}$, we assume that the initial position of the phases and their interface is given in a way that provides solutions to these problems. When generating $\mathcal{G}_{h,n}$ for $n = 1, \ldots, \frac{T}{\Delta t}$, the solutions to these problems are an essential part of the alignment algorithm.

2.4.3 Computation of the phase of given point

We use a discontinuous marker function to identify the phase of points. Suppose that on some grid $\mathcal{G}_{h,n}$ we have marked already each node in phase i with value of the marker function i. The nodes on $\partial\Omega$ belong to the background phase 0, while the nodes on Γ are marked with a code that is different from any of the phase indices, a code that uniquely identifies the interface. We extend the marker function over the elements and the faces/edges, which gives us the phase marker at time $t = t_n$. Because of the alignment of $\mathcal{G}_{h,n}$ with the

interface at time $t = t_n$, as well as the requirement that no element has three interface nodes, each element has either all nodes in the same phase, or some, but not all, nodes on the interface. However, no element has nodes in different phases. Also, no edge has nodes in different phases, and only interface edges have both nodes on the interface. Thus for each element we find a node that is not on the interface, and the value of the phase marker at this node is extended as the value of the phase marker in the interior of the element. Moreover, if an edge has a node that isn't on Γ, then the phase of this node becomes the phase of the edge. An edge with both nodes on the interface is marked with the code of the interface.

In the first step of the time-marching algorithm described in section 2.3.2, we advect the nodes in $\mathcal{G}_{h,n}$. In doing so, the positions of the nodes are changed, but the values of the phase marker for nodes, edges, and elements remain unchanged. This gives us an approximation of the phase marker at time $t = t_{n+1}$. All we need to do to determine the phase of a point is to find an element, edge, or a node in the *advected* $\mathcal{G}_{h,n}$ mesh that is incident with the point, and take its phase marker. Since the points whose phase we actually need to compute are the reference positions of the nodes, we can limit our search to the elements containing the node, since the movement of the grid nodes over one timestep is small.

2.4.4 Computation of the projection onto the interface

The solution of this problem is a bit more involved. The interface nodes and the interface faces in $\mathcal{G}_{h,n}$ form a piecewise linear approximation $\Gamma_{h,n}^n$ of the position of the interface at time $t = t_n$. After advecting the nodes in $\mathcal{G}_{h,n}$, the new positions of the interface nodes form a piecewise linear approximation $\Gamma_{h,n}^{n+1}$ of the interface at time $t = t_{n+1}$.

We could, of course, project points onto the piecewise linear curve directly. This has shown, however, to lead to instabilities, thus a smoother approximation is needed. A solution leaning towards the opposite extreme is to use a cubic spline interpolation through the interface nodes (see, for example, [50]) and project onto this curve. We chose a compromise between the two approaches – the interface is approximated with a smooth curve locally, avoiding the computations needed for the construction of a spline, while providing enough smoothness to keep the algorithm stable.

Let P be any point in Ω. Given the approximation $\Gamma_{h,n}^{n+1}$ of the interface, we want to construct the point P' on Γ^{n+1} that is closest to P. The projection procedure relies on simple geometric arguments (see Figure 16):

- Compute the distance from P to every node on $\Gamma_{h,n}^{n+1}$. Find node A in $\Gamma_{h,n}^{n+1}$ that is closest to P.

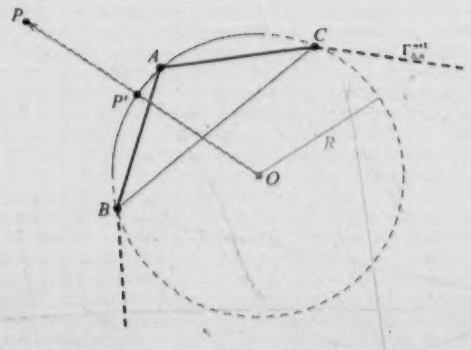

Figure 16: Construction of the projection P' of a point P onto the interface by approximating Γ locally with a circle.

- Find nodes B and C – these are the nodes we encounter right after and right before A, if we traverse Γ^{n+1} going counter-clockwise around the bubble Ω_1.
- Construct point O as the circumcenter of $\triangle ABC$. Also compute the radius R of the circumscribed circle.
- Construct point P' as the point of intersection of the circumscribed circle of $\triangle ABC$ and the line OP. We compute its position as the point on the ray $\overrightarrow{OP}$, which is at distance R from O.

It is clear from this construction that we approximate the interface locally with a circle. Since this approximation is local, it is also computationally inexpensive. Moreover, a great advantage of using a circle is the fact that this approximation is exact when Γ^{n+1} is a circle. This allows us to resolve exactly the problem of a circular drop left to relax under the force of surface tension alone. Most Eulerian and even some Lagrangian and ALE formulations using piecewise linear approximations fail this simple test, because of the appearance of parasitic currents and unphysical oscillations of the interface.

2.4.5 Computation of the surface tension

The surface integral along Γ appearing in the right-hand side of equations (2.19) and (2.24) is actually computed along the approximation $\Gamma_{h,n+1}^{n+1}$ of the interface embedded in the computational grid $\mathcal{G}_{h,n+1}$:

$$\frac{1}{We}\int_{\Gamma}\varkappa\mathbf{v}_h\cdot\mathbf{n}\,ds \approx \frac{1}{We}\sum_{f_i\in\mathcal{F}_{h,n+1}\cap\Gamma_{h,n+1}^{n+1}}\int_{f_i}\varkappa\mathbf{v}_h\cdot\mathbf{n}\,ds. \tag{2.25}$$

The unit normal vector appearing in the integrand is taken to be an approximation of the normal vector to Γ^{n+1} and not the normal vector to $\Gamma^{n+1}_{h,n+1}$. The curvature $\varkappa$ and the normal vector $\mathbf{n}$ are computed at each interface node in $\mathcal{N}_{h,n+1} \cap \Gamma^{n+1}_{h,n+1}$ from the same local circle approximation that we used for the projection on the interface. Thus at each node $\mathbf{x}_i^{n+1} \in \mathcal{N}_{h,n+1} \cap \Gamma^{n+1}_{h,n+1}$ the curvature is taken as the reciprocal of the radius of the approximating circle, and the unit normal is taken as the unit normal to the circle pointing out of Ω_1.

We use a one point quadrature formula for the integral along each interface face f_i. The value of $\varkappa$ at the midpoint m_i of the face is simply the average of the curvatures at the two nodes of f_i, while $\mathbf{n}$ is taken as the average of the normal vectors at the face's nodes, then normalized to unit length. Thus the approximation of the surface integral is

$$\int_{f_i} \varkappa \mathbf{v}_h \cdot \mathbf{n} \, ds \approx \frac{1}{2} \left(\varkappa_{i,0} + \varkappa_{i,1} \right) \mathbf{v}_h(m_i) \cdot \left(\frac{\mathbf{n}_{i,0} + \mathbf{n}_{i,1}}{\| \mathbf{n}_{i,0} + \mathbf{n}_{i,1} \|} \right) |f_i| \,. \tag{2.26}$$

Note that if the interface is a circle, the sum of the two nodal normal vectors is a vector perpendicular to f_i. Also in this case, the nodal curvatures are equal. Therefore, the integral computed this way actually produces the weak gradient of a pressure field, which is constant in each fluid and exhibits an appropriate jump at the interface. This means that the projection of the surface tension onto the divergence-free space $\mathbf{V}_{h,n+1}$ is zero and, therefore, the problem with circular bubble relaxing under surface tension but no gravity will be resolved exactly. It is easy to verify that this is not true if we use a more accurate higher degree approximation of the surface integral. Higher smoothness of the approximation would make it in a way "incompatible" with the solenoidal projection that we are using, since the piecewise constant pressure would no longer be able to neutralize the more accurate surface tension, and, as a result, parasitic velocities would appear in $\mathbf{u}^{n+1}_{h,n+1}$.

2.4.6 Numerical validation

The proposed algorithm was extensively tested on a number of validation problems. The lid-driven cavity and the rising bubble examples were computed with the ALE approach for velocity transfer between two subsequent grids. The rest of the validation examples were computed both ways. In all cases the results from the ALE and the interpolation approaches were almost identical.

Lid-driven cavity flow. The first test was aimed at validating the grid alignment algorithm. We ran two simulations of the well known lid-driven

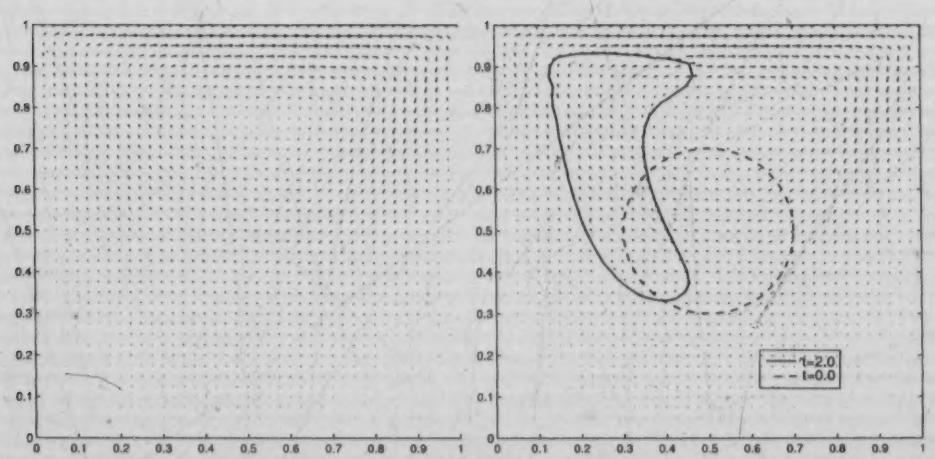

Figure 17: Comparison of simulation results of lid-driven cavity flow with (right) and without (left) a free moving interface. $Re = 40$, $\Delta t = 0.005$, grid 40×40, no surface tension.

cavity problem. In the first one, we had only one fluid and no moving interfaces. In the second simulation, we placed an artificial circular interface separating two regions occupied by the same fluid ($\rho_1/\rho_0 = 1$ and $\mu_1/\mu_0 = 1$). The circle was initially centered at the geometric center of the cavity and had a radius $r = 0.2$. In this simulation, there was no gravity and no surface tension. We used $Re = 40$ and $\Delta t = 0.005$. We ran simulations on 40×40 grids in $[0,1] \times [0,1]$. Both simulations were terminated at $T = 2.0$, at which time the interface had undergone a severe deformation. Figure 17 shows that the velocity fields resulting from the two simulations are essentially the same. The positions of the moving interface at times t=0 and t=2 are shown on the right graph.

Convergence test. We ran a series of tests with an analytic solution with and without a moving interface to investigate the influence of the grid alignment on the convergence. In all tests we set $\rho_1 = \rho_0 = 1$, $\mu_1 = \mu_0 = 1$, $Re = 100$, and there was no surface tension. The tests were performed with the following non-trivial analytic solution

$$\begin{aligned} u &= \sin(x)\sin(y+t), \\ v &= \cos(x)\cos(y+t), \\ p &= \cos(x)\sin(y+t), \end{aligned}$$

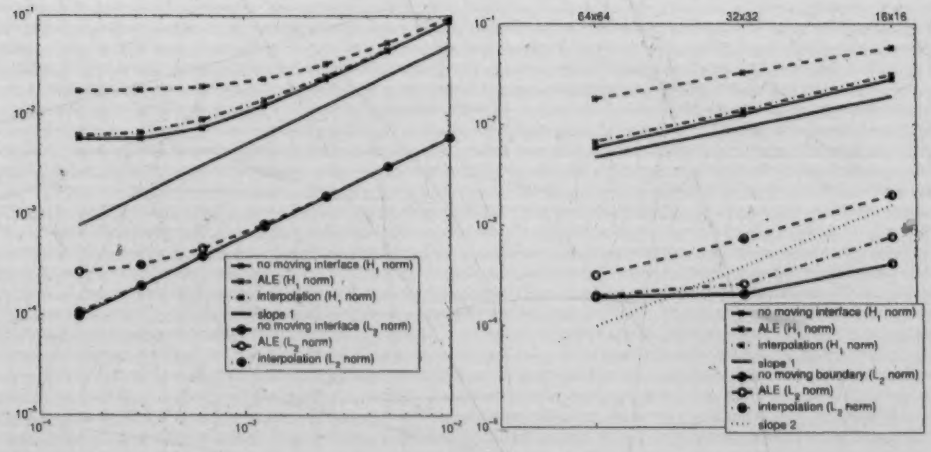

Figure 18: Convergence of $\tilde{\mathbf{u}}$ in time (left), grid 64×64 and space (right), $\Delta t = 0.0003125$.

with an appropriate force added to the right-hand side of the governing equations. The cases with a moving interface were solved in two ways – once using interpolation to move data between grids and a second time using the Lagrangian technique described in section 2.3.3. Figure 18 presents the convergence in space and time of $\tilde{\mathbf{u}}$ in L^2- and H^1-norms. It is clear that the presence of the moving intrface does not disturb the convergence. Using interpolation seems to be more accurate in this test. However, in the case of significantly different viscosities, the ALE scheme may yield better results.

Static circular drop. Consider a spherical (circular in two dimensions) fluid particle immersed in another fluid and subject only to non-zero surface tension (without gravity). Theoretically, the shape of the interface will not deviate from circular, and the velocity field will remain constant 0 at all times. This test is impossible to resolve exactly on general Eulerian grids due to parasitic velocities near the interface, which result from the fact that some elements are crossed by the interface. For strong surface tension and sufficiently long integration in time, a significant loss of mass may be accumulated. On the other hand, the exact solution consists of a constant velocity equal to zero and a piecewise constant pressure in each phase. If the numerical scheme is consistent and optimal, it should resolve this problem exactly, because the elements used provide a piecewise linear approximation for the velocity and a piecewise constant approximation for the pressure. The present method is exact for this problem. We solved it in the square $[-1,1]\times[-1,1]$ with parameters $\rho_1/\rho_0 = 1$, $\mu_1/\mu_0 = 1$, $Re = 10$. We used 40×40 and 80×80 grids

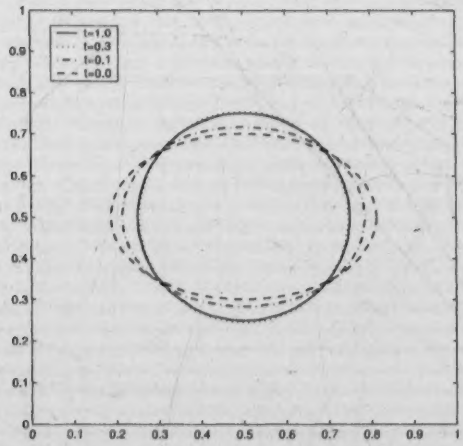

Figure 19: Relaxation of a two-dimensional elliptical particle. $We = 1.0$, $\Delta t = 0.0025$, grid 40×40.

and different values of the surface tension coefficient $We = 0.2, 1.0, 5.0$. We started with timestep $\Delta t = 0.01$, but decreased it to 0.005 and then again to 0.0025 as the surface tension coefficient increased and the grid became finer. The need for the smaller timestep to maintain the stability of the solution is due to the explicit treatment of the advection terms and the nonlinearity in the surface tension. All simulations yielded the exact final solution.

The next example is for the advection of a particle with a constant velocity $\mathbf{u} = (1, 0)$. The boundary and initial conditions were all set to $(1, 0)$, $\rho_1/\rho_0 = 1$, $\mu_1/\mu_0 = 1$, $Re = 10$, and $We = 0.2, 1.0, 5.0$. Theoretically, the shape and the velocity of the drop must not change as it moves. This time the domain was $[-1, 1] \times [-0.5, 0.5]$ and the circular particle was centered initially at $(-0.5, 0)$. We used a 48×24 grid, the time step was set to $\Delta t = 0.00125$, and the tests ran until $T = 1.0$. The numerical results show that the interface indeed remains circular and the velocity deviates only slightly, the deviation being of the order of 10^{-4}. This deviation is a result of the transfer of data between grids taking place at every time step. It was observed, however, that nodes behind the moving particle restore the exact velocity as soon as the drop passes by.

Relaxation of a drop of a simple shape. This is a classical test for free boundary methods, in which the initial shape of the particle slightly deviates from circular and the gravity is set to zero. The problem was solved in the square $[0, 1] \times [0, 1]$, with parameters $\rho_1/\rho_0 = 1$, $\mu_1/\mu_0 = 1$, $Re = 10$, $We = 1.0$ using a grid of 40×40 nodes and a timestep $\Delta t = 0.0025$. Initially, the drop

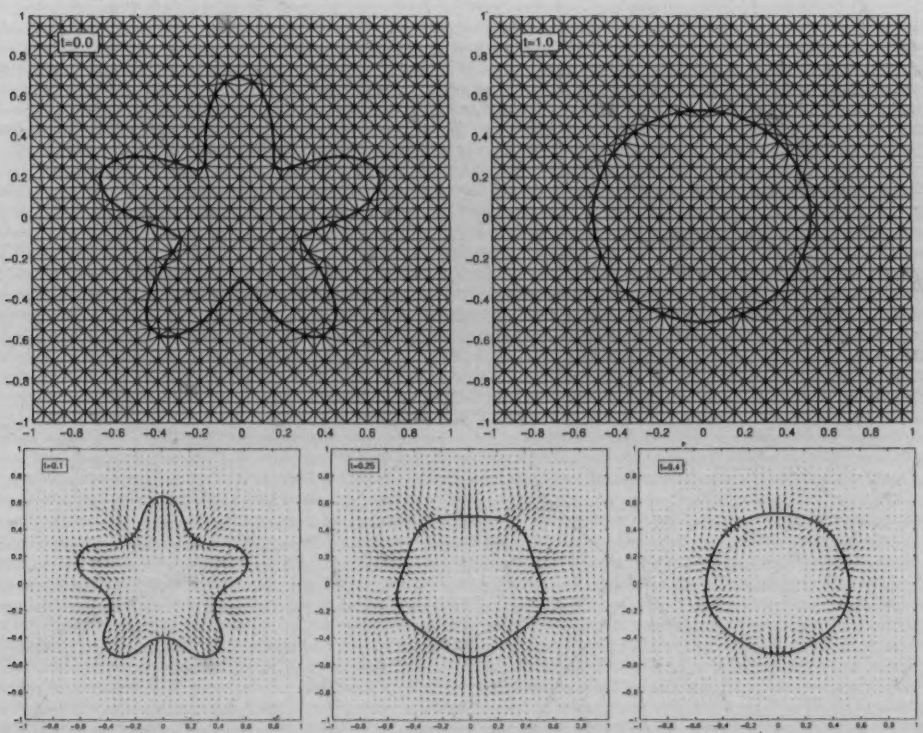

Figure 20: Relaxation of a two-dimensional star-shaped particle. $We = 1.0$, $\Delta t = 0.005$, grid 40×40.

is elliptical with axes 5/16 and 1/5, i.e. it has the same area as a circle with a radius of 1/4. The drop becomes circular after a time period of 0.3 (see Figure 19).

Relaxation of a drop of a complex shape. The next test is to compute the relaxation of a drop of initially complex shape to a circle. The initial star-like curve is given by

$$r = 0.2 \sin(5\theta) - 0.5, \tag{2.27}$$

as suggested in [51]. The problem was solved in $[-1, 1] \times [-1, 1]$ on a 40×40 grid with $\Delta t = 0.005$ until time $T = 1.0$. The values of the parameters are the same as in the previous example. The drop assumed the static shape at approximately $T = 0.4$. In Figure 20, we present the initial and final form of the drop together with the corresponding aligned grids, as well as the shape

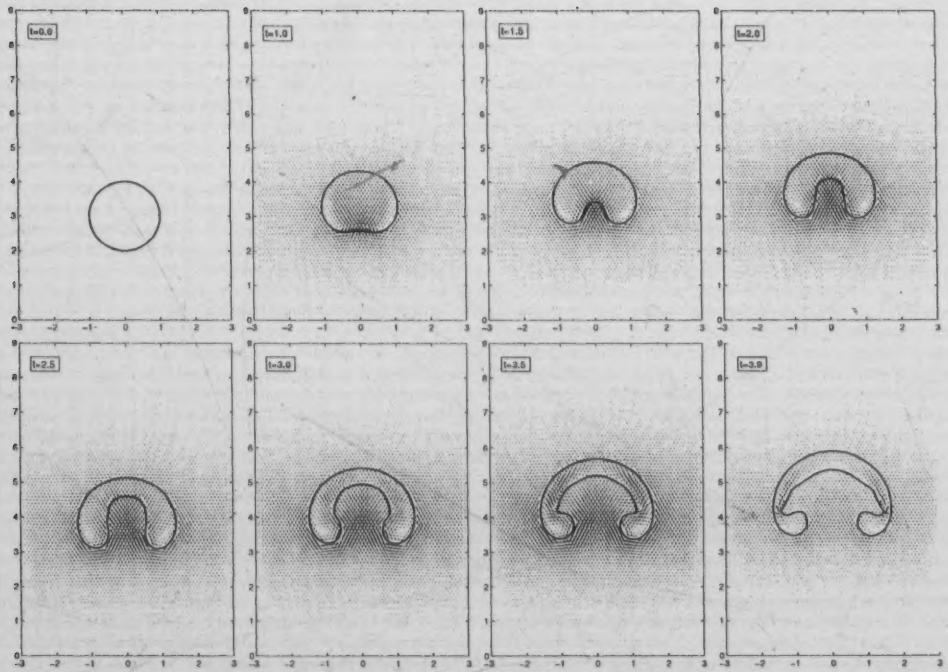

Figure 21: Simulation of the motion of a two-dimensional gas bubble in a liquid. $Re = 1000$, $Fr = 1$, $We = 200$, $\rho_1/\rho_0 = 1/816$, $\mu_1/\mu_0 = 1/64$, $h = 0.1$, $\Delta t = 0.0005$.

of the interface with the velocity field at times $T = 0.1, 0.25, 0.4$.

Rising bubble. In the last test we examine the motion of a gas bubble immersed in a heavier liquid. Under the force of gravity, the bubble starts to rise and, depending on the strength of its surface tension, either assumes a static shape and a constant velocity, or develops a pinch-off. We simulated conditions similar to the experimental investigation in [91]. The parameters are $Re = 1000$, $Fr = 1$, $We = 200$, $\rho_1/\rho_0 = 1/816$, and $\mu_1/\mu_0 = 1/64$. Initially the bubble has a circular shape with a radius equal to 1, and is centered at $(0, 3)$. The computational domain is $[-3, 3] \times [0, 9]$, and the problem is discretized on a mesh with a grid size $h = 0.1$. The time step is $\Delta t = 0.000125$. The evolution of the bubble is presented in Figure 21.

In the experiments of [91], the pinch-off started at $t = 4.42$. Since the pinch-off cannot be simulated with the present algorithm, we only compare the results up until the pinch-off starts to develop. The shapes we obtained correlate well with the experimental results (see Figure 22).

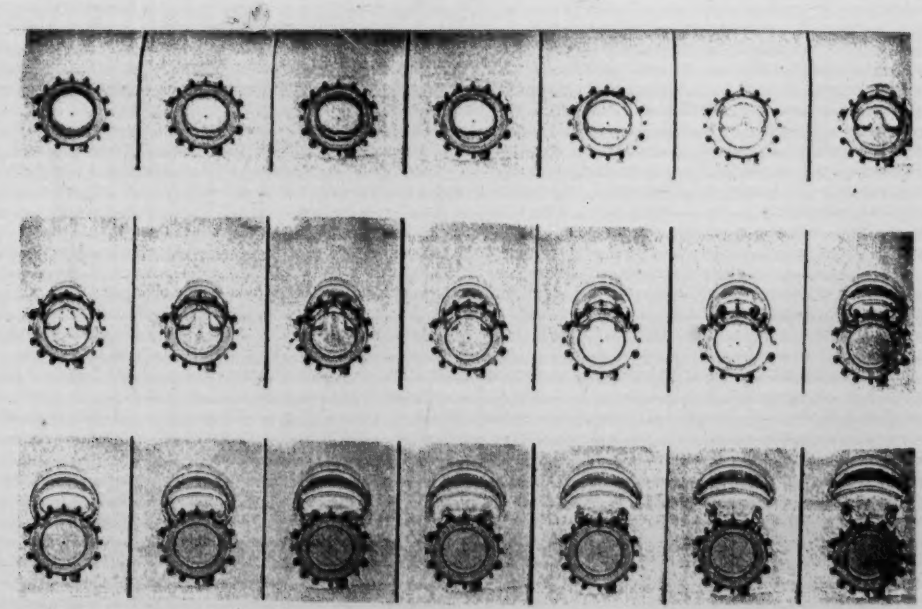

Figure 22: An experimental result for a rising two-dimensional bubble from [91]. The time between two subsequent frames is about 0.25.

2.5 Grid alignment in three spatial dimensions

In this section we discuss the case of $N = 3$. If we examine carefully the steps in the alignment algorithm proposed in section 2.4.1, we notice that each step can be generalized to three dimensions directly. However, there are situations in three dimensions that are not possible in two dimensions. The algorithm, in its current form, is not able to handle such situations; for this reason, parts of it need to be modified.

2.5.1 Specific difficulties in three dimensions

The first problem is related to the appearance of elements that have all their nodes on the interface after the alignment is done. In step 5 of the alignment algorithm, we find a node that can be removed from the interface without introducing intersected edges. This is not always possible in three dimensions – an element can have four nodes on the interface with each node having neighbors on both sides of Γ.

Suppose that we have applied the first four steps of the algorithm in section 2.4.1 in three dimensions, and that in step 5 we have encountered an element with 4 interface nodes. Obviously, it is not possible for this element

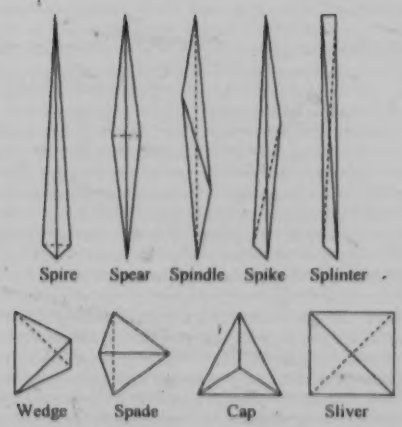

Figure 23: Tetrahedral elements of poor quality.

to have all 4 faces on the interface. It is not possible for exactly one face to be on Γ either. If the interface faces are exactly 3, then their common node can be removed safely from the interface. Indeed, it is easy to see that the assumption that this node has neighbors on both sides of the interface leads to the conclusion that there is either an edge crossed by Γ or some elements must be inverted. If, however, the element has 2 faces on the interface, then it is possible for each node to have neighbors in both phases. In this case, it is impossible to decide which two faces are on the interface, and equivalently, we cannot decide to which phase the element belongs. Moreover, the two faces of the element that are not on the interface have all their nodes on Γ, so simply counting the interface nodes of a face is no longer representative for whether this face is on Γ or not.

One way to deal with this situation is to evaluate the phase marker at the centroid of the element and prescribe this value as the phase of the element. Once we know the phases of all elements, we can evaluate the phase marker of each face by looking at the phases of the two elements sharing the face. If the phases are the same, then the face belongs to the same phase; if they are different, then the face is on Γ. Soon we will see that this solution is not acceptable, since elements with four interface nodes often have very poor quality, which is impossible to correct.

The second major difficulty that we meet in three dimensions is the extremely poor quality of the grid after alignment. The reason why this is a major problem in three, but not in two dimensions is that in three dimensions, the elements have many more ways in which they can get distorted. Figure 23 is taken from [27]; it illustrates the vast variety of poorly shaped

tetrahedral elements. In two dimensions it was enough to move nodes near the interface to the centroids of their corresponding polygons to improve the quality of the grid. This procedure, however, cannot fix elements in three dimensions that have all their nodes on Γ, since we are not allowed to remove these nodes from the interface. Furthermore, in two dimensions we can change the quality of an element to any value by moving a single node, while in three dimension the maximum quality we can achieve the same way is limited by the shape of the fixed face.

Based on these considerations, we modify the algorithm as follows. In step 3, we need a method for selecting nodes to move onto the interface that guarantees that no element will have 4 nodes on Γ, since we are not always able to fix this problem later. In addition, after checking the integrity of the interface and before the volume adjustment, we add a step in which we monitor and, if necessary, improve the quality of the faces on Γ. This will help us to achieve better quality of the elements when we improve the quality of the grid after adjusting the volume.

2.5.2 Alignment algorithm

Our setup here is the same as in the two-dimensional case (section 2.4.1). The three-dimensional algorithm proceeds with the following steps.

1. Create lists of intersected edges, pending nodes, and elements with four pending nodes. The lists of intersected edges $\mathcal{X}_h$ and pending nodes $\mathcal{P}_h$ are exactly the same as in the two-dimensional case. Once we have $\mathcal{P}_h$, we create a list $\mathcal{Q}_h \subset \mathcal{T}_h$ of elements that have four nodes in $\mathcal{P}_h$. These elements are the only ones that potentially can become elements with four nodes on Γ.

2. Sort nodes. This step is the same as in two dimensions, except that this time we sort pending nodes in the opposite direction – we put nodes whose distance to the interface is smallest at the end of the list. See the next step and section 2.5.3 for further discussion.

3. Move nodes onto the interface. The essential part of this step is actually choosing which nodes to move onto Γ. We select some pending nodes in such a way that at least one node is selected from each edge in $\mathcal{X}_h$, while at least one node from each element in $\mathcal{Q}_h$ is not selected. The details are presented in section 2.5.3. The selected nodes are then projected onto Γ and marked as interface nodes.

4. Nodes consistency. This step is the same as in two dimensions. Note that we don't need to check if elements have 4 nodes on Γ, because step 3 guarantees that they don't.

5. Interface consistency. As in the two-dimensional case, we count the number of interface nodes adjacent to each interface node, but this time we require that this number is greater than or equal to 3. If an interface node has 2 or less neighbors on Γ, then we remove it by placing it in the centroid of its polyhedron. In this case, we also set the node's marker function equal to the phase of its neighbors.

After all disconnected nodes are removed, we count the number of interface faces containing each interface edge. If Γ is closed, then this number must be exactly 2 for all edges on Γ. If it is less than 2, then there is a tear in the surface. A number greater than 2 indicates that the surface is crossing itself at this edge. In both cases the current implementation stops. Otherwise, handling of topology change in the interface can be done in this place of the algorithm.

6. Quality of the interface grid. We check the quality of each face on Γ. If the minimal quality is below a prescribed threshold, we apply a version of Laplacian smoothing geared for surface grids. For each node, we take only those neighbors that are on Γ, add their coordinates, and divide by their number. The projection of the resulting point on Γ is the new position of the node.

7. Volume adjustment. This step is the same as in two dimensions.

8. Quality of the grid. After the volume adjustment, elements near the interface often have very poor quality, and Laplacian smoothing has proved incapable of correcting this problem. Instead, we use an optimization-based local smoother presented in [28]. Obviously, it is not necessary to apply this smoother on the whole grid, since most nodes are in their reference positions, and we assume that the reference grid has acceptable quality. On the other hand, it is not enough to smooth only the nodes directly connected to the interface, since their polyhedrons are deformed. If we allow a few layers of nodes to move, after only a few smoothing passes we achieve a significant improvement in the quality of the elements. In the current implementation, we allow 3 layers of nodes to move; these are all nodes that can be connected to the interface with a chain of no more than 3 edges. The number of layers

can be adjusted according to the desired balance between grid quality and computational effort.

2.5.3 Algorithm for selecting interface nodes

The nodes can be identified by their indices. Let P be the set of nonnegative integers, equal to the indices of the pending nodes in $\mathcal{P}_h$. Each edge can be identified by its two nodes, so we define $X \subset P^2$ as the set of all pairs of integers from P corresponding to the edges in $\mathcal{X}_h$. Similarly, elements are identified by their four nodes, so we define $Q \subset P^4$ as the set of all combinations of 4 integers from P corresponding to the elements in $\mathcal{Q}_h$. We define the problem as follows: find a set $P^+ \subset P$ such that each pair in X contains at least one number from P^+, while each 4-tuple in Q contains at least one number from $P^- \equiv P \setminus P^+$. The nodes in the corresponding $\mathcal{P}_h^+$ will be projected onto the interface, while the remaining nodes will be left in their reference positions.

We have developed a recursive algorithm for finding a solution, which uses five subroutines. Auxiliary array is used to store information for each number in P regarding whether it is currently in P^+, P^-, or neither, and on which level of recursion this number was handled.

The main routine

1. Select a number from P. If there are no numbers left in P, then fail.
2. Try recursion with the selected number at level 2.
3. If the recursion is successful, then we are done.
4. Otherwise, move the selected number from P to P^+ and mark it as handled on level 1. Go back to step 1.

The routine performing the recursion

Given a number and level of recursion:

1. Move the given number from P to P^- and mark it as handled at the current level.
2. Check pairs in X at the current level.
3. Check 4-tuples in Q at the current level.
4. If either check fails, then clean the current level and return indicating failure.
5. If all pairs in X are "done," then return indicating success.
6. If either check moved any nodes, then go back to step 2.
7. Select a number from P. If there are no numbers left in P, then fail.
8. Try the recursion with the selected number at the next level.
9. If the recursion is successful, then we are done.

10. Otherwise, move the selected number from P to P^+ and mark it as handled on the current level. Go back to step 7.

The routine checking the pairs in X

Given level of recursion, for each pair $(p_0, p_1) \in X$ do steps 1-4.

1. If $p_0 \in P^-$ and $p_1 \in P$, then move p_1 from P to P^+ and mark it as handled at the current level.

2. If $p_0 \in P$ and $p_1 \in P^-$, then move p_0 from P to P^+ and mark it as handled at the current level.

3. If $p_0 \in P^-$ and $p_1 \in P^-$, then return failure.

4. If $p_0 \in P^+$ or $p_1 \in P^+$, then count this pair as "done."

5. If all pairs are "done" and no numbers have been moved, then return success indicating that all pairs are "done."

6. Otherwise, return success indicating whether any nodes have been moved or not.

The routine checking the 4-tuples in Q

Given level of recursion, for each 4-tuple $q \in Q$ do steps 1-3.

1. If any number in q is in P^-, then continue with the next q.

2. If 4 numbers in q are in P^+, then return failure.

3. If 3 numbers in q are in P^+, then move the fourth number from P to P^- and mark it as handled at the current level.

4. Return success indicating whether any nodes have been moved or not.

The routine that cleans up a level simply moves all nodes handled at the current or any higher level back into P and marks them as not handled.

The general idea of this algorithm is to start with empty P^+ and P^-. Every number $p \in P$ is first placed in P^-. Now all pairs in X containing p must have all their other numbers moved to P^+. Next, there may be 4-tuples in Q that have 3 numbers in P^+ – the fourth number must now be moved to P^-. If we have moved any numbers in P^-, then we need to check X again, then possibly check Q again, and so on. We keep checking X and Q until we either find a problem, or there is no reason to move nodes anymore. A problem would be a pair with both numbers in P^-, or a 4-tuple with all numbers in P^+. If this happens, then all numbers that have been moved because of p must be moved back to P, while p itself is placed in P^+. Then the loop closes, and a new number $p \in P$ is processed the same way. The algorithm stops when all pairs in X contain at least one number in P^+. At this time, if there are any numbers left in P, they are placed in P^-.

In many situations, the number of possible divisions of P into P^+ and P^- that meet our needs is more than 1. The solution produced by the algorithm, and also its speed, depends on the specific order in which we process numbers. By placing numbers in P^- first, and moving numbers to P^+ only if we have to, we ensure that the produced solution will be the one with the least amount of numbers in P^+. For the grid alignment, this means that we first try to see if we can align the grid without moving a given node, and moving it only if there is no other way. Naturally, we should try not moving the nodes that are farthest from the interface first, hence in step 2 of the alignment algorithm we sort the pending nodes the way we do. Also, having the smallest amount of interface nodes means that there will be more nodes that we are allowed to move for the purpose of improving the quality of the grid. On the other hand, having more nodes on the interface may improve its approximation. Such improvement, however, can be at most very small, as the order of approximation is limited by the resolution of the reference grid.

It is clear that if a solution of our problem exists, then this algorithm will find it, and that the produced splitting of P meets our needs. Although we don't have a rigorous proof that the problem will always have a solution, this algorithm has produced a valid solution in all simulations that we have run so far. In case the algorithm ever fails, we are prepared to use the old algorithm as a last resort and allow elements with 4 interface nodes for one timestep. The possible trade-off for this is the poor quality of the grid, which we may not be able to fix.

2.5.4 Approximation of the interface

The computation of the extension of the phase marker from its values at the nodes to edges, faces, and elements is the same as is the two-dimensional case. The computation of the projection of a point onto the interfaces, as well as that of the mean curvature and outer normal needed for the surface tension, is again done by smoothing Γ locally, this time with a sphere.

The sphere approximating Γ in the vicinity of an interface node P is constructed using the node together with all of its immediate neighbors on Γ. Obviously, each neighbor is connected to P by an interface edge. We construct the plane passing through the midpoint of and perpendicular to the edge. If P and all of its neighbors lie on an exact sphere, then all these planes will pass through the center of this sphere. Algebraically, the equations of these planes form a linear system

$$A\mathbf{c} = F,$$

where the unknown $\mathbf{c} \in \mathbb{R}^3$ is the center of the sphere, and each row in the

matrix A contains the coordinates of the tangent vector to one of the edges. The rank of A is at least 2, since all edges contain the same node, and is equal to 2 if and only if P and all of its neighbors are on the same plane. The latter is due to the fact that there are at least 3 rows in A, which is guaranteed by step 5 in the alignment algorithm (section 2.5.2). Of course, generally, there are more than 3 equations, and, unless the nodes are on a sphere, the system is inconsistent. That's why we solve the least-squares problem

$$A^T A\mathbf{c} = A^T F$$

instead. The solution is unique if the nodes are not on the same plane. Then the mean curvature is approximated by the reciprocal of the distance from P to the solution $\mathbf{c}$, while the normal is taken to be collinear to the line through P and $\mathbf{c}$. If $\det A^T A = 0$, the curvature is 0, and the normal is perpendicular to the plane passing through P and all of its neighbors on Γ.

Similarly to the two-dimensional case (section 2.4.5), we use one point quadrature formula to compute the surface tension integral on each face. The value of the curvature at the centroid is the average of the nodal values of $\varkappa$. The average of the nodal normal vectors, however, is not a good approximation of the normal vector to Γ, since it is not exact for a sphere. Suppose that all interface nodes are on the same sphere with center $\mathbf{c}$ and radius R. The normal vectors at the three nodes $\mathbf{x}_{i,j}, j \in \{0,1,2\}$, of given face f_i are $\mathbf{n}_{i,j} = \frac{1}{R}(\mathbf{x}_{i,j} - \mathbf{c})$. The average of these three normals, in general, does not give a vector perpendicular to f_i, as it did in two dimensions. In three dimensions, the vector perpendicular to the face is collinear with the line through $\mathbf{c}$ and the face's circumcenter $\mathbf{c}'$ (not the centroid). Obviously, if $(\gamma_0, \gamma_1, \gamma_2)$ are the homogeneous barycentric coordinates of $\mathbf{c}'$, i.e.

$$\mathbf{c}' = \gamma_0\mathbf{x}_{i,0} + \gamma_1\mathbf{x}_{i,1} + \gamma_2\mathbf{x}_{i,2},$$

then

$$\mathbf{c}' - \mathbf{c} = \gamma_0(\mathbf{x}_{i,0} - \mathbf{c}) + \gamma_1(\mathbf{x}_{i,1} - \mathbf{c}) + \gamma_2(\mathbf{x}_{i,2} - \mathbf{c}),$$

and, therefore, we approximate the normal to Γ by taking a linear combination of the three nodal normal vectors with coefficients γ_j. The surface tension integral on f_i is evaluated by the formula

$$\int_{f_i} \varkappa \mathbf{v}_h \cdot \mathbf{n}\, ds \approx \frac{1}{3}\left(\varkappa_{i,0} + \varkappa_{i,1} + \varkappa_{i,2}\right) \mathbf{v}_h(m_i) \cdot \left(\frac{\gamma_0\mathbf{n}_{i,0} + \gamma_1\mathbf{n}_{i,1} + \gamma_2\mathbf{n}_{i,2}}{\|\gamma_0\mathbf{n}_{i,0} + \gamma_1\mathbf{n}_{i,1} + \gamma_2\mathbf{n}_{i,2}\|}\right) |f_i|\,.$$

This approximation meets our requirement to be exact if all interface nodes

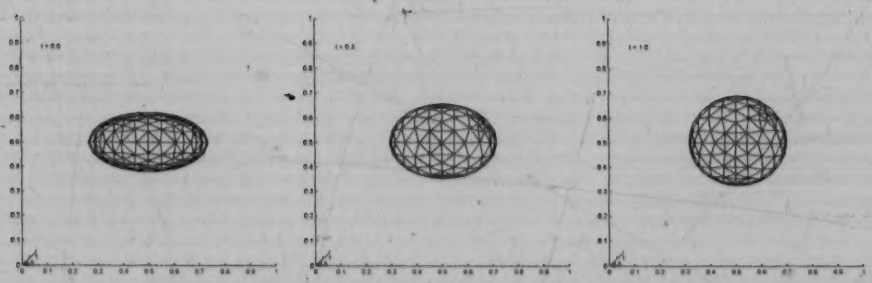

Figure 24: Relaxation of an ellipsoidal particle.

are on the same sphere. On the other hand, we foresee its potential problems in situations where the interface contains a saddle point, since a sphere is not an adequate approximation of such surface. These situations are more likely to appear in simulations of free surface flows, where the moving surface is not closed. In our simulations of bubbly flows, this simple approximation has shown to be adequate.

2.5.5 Numerical validation

All tests were performed using the ALE approach for handling the transfer of data between grids. We used the preconditioned Lagrange multipliers for the solenoidal projection as described in section 1.6.4.

Static spherical particle. In this test we simulated the relaxation of an initially spherical particle placed in a quiescent fluid. There was no gravity and the only force present was the surface tension. The domain was the cube $[0,1]\times[0,1]\times[0,1]$, and the initial interface was a sphere with radius 0.25 and center $(0.5, 0.5, 0.5)$. The Reynolds and Weber numbers were set to $Re = 100$ and $We = 0.2, 0.5, 1.0$, while the ratios of density and viscosity were set to $\rho_1/\rho_0 = \mu_1/\mu_0 = 1$. All test produced the exact solution, where the velocity remains constant 0 and the interface does not deviate from the initial shape.

Relaxation of ellipsoidal particle. An initially ellipsoidal particle is left to relax under the force of surface tension until it assumes spherical shape. The fluid velocity should gradually diminish to 0. The computational domain $[0,1]\times[0,1]\times[0,1]$ was discretized with a $16\times16\times16$ grid. The initial shape of the bubble obeys the equation

$$\frac{(x-c_x)^2}{a^2} + \frac{(y-c_y)^2}{b^2} + \frac{(z-c_z)^2}{c^2} = 1,$$

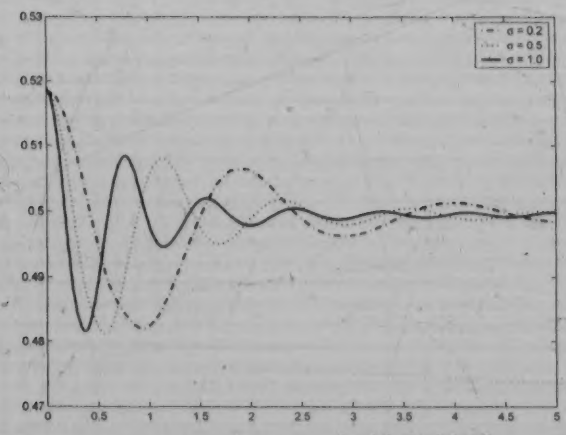

Figure 25: The frequency of an oscillating bubble.

where $\mathbf{c} = (0.5, 0.5, 0.5)$ and $a = b = 0.25$, $c = 0.128$. The physical parameters of the problem were set to $Re = 10, We = 10, \rho_0/\rho_1 = 1, \mu_0/\mu_1 = 1$. The evolution of the shape of the particle is illustrated in Figure 24. It is clear that the particle undergoes the expected behavior.

Drop oscillations. The accuracy of the proposed technique was verified by comparison with a classical solution of a moving interface problem from [49, p.473], where potential theory was used to derive the solution. We let an ellipsoidal particle relax and measure the frequency of oscillation. The initial shape of the bubble has $a = b = 0.55$, $c \approx 0.4132$ and its volume equals the volume of a sphere with radius 0.5. The computational domain this time is $[-1, 1] \times [-1, 1] \times [-1, 1]$. The grid we used was $16 \times 16 \times 16$. The densities were chosen to be $\rho_0 = 1, \rho_1 = 0.01$, while the viscosities were set to $\mu_0 = 0.01, \mu_1 = 0.0002$. We ran the test for $\sigma = 0.2, 0.5, 1.0$. Figure 25 presents the evolution of the position of the point of intersection of the x-axis with the interface. The frequencies we obtained were 1.9, 1.14, and 0.77 respectively, which compare well with the theoretically predicted values of 1.76, 1.11, and 0.79.

Rising bubble. The last example is a simulation of a bubble rising in a heavier fluid due to gravity. Depending on the values of the Froude and the Weber numbers, the bubble will either assume a steady shape and a constant vertical speed, or undergo pinch-off or other instabilities. Our simulation was performed in the domain $[-2.5, 2.5] \times [-2.5, 2.5] \times [-1.5, 6]$, dis-

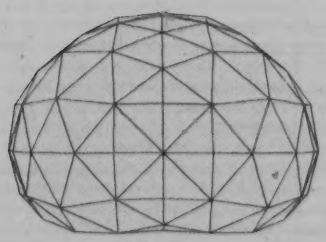

Figure 26: The computed shape of a bubble steadily rising in heavier fluid.

cretized with a $16 \times 16 \times 24$ grid, where the initially spherical bubble was centered at the origin and had radius 1. The physical parameters were set to $Re = 9.8, We = 7.6, Fr = 0.76, \rho_1/\rho_0 = 0.0011, \mu_1/\mu_0 = 0.0085$. The shape presented in Figure 26 is obtained at $t = 1.15$ of the computation, at which point the bubble has not yet assumed its steady shape. Further computation leads to instability due to the inability of the coarse grid to resolve adequately the large deformation developing in the bubble. Unfortunately, we have not been able to use finer grids due to the exceedingly long computational times. As a result, we are not able to compare our results with the steady shape obtained experimentally in [44], which was our initial intention. However, the comparison with the shape obtained in a numerical simulation presented in [77] at $t = 1.2$ was found satisfactory.

2.6 Conclusion

In the second part of our study, we dealt with an application of the projection method devised in the first part to simulations of flows with moving capillary surfaces. We started by formulating the problem in appropriate nondimensional variables and presented the proper spatial and temporal discretizations. The most important feature of the technique developed here is the fact that the grid is modified on each timestep to reflect the changing position of the moving interface. The purpose of such modification is to align the grid with the interface. The alignment procedure also keeps the connectivity unchanged. Since computations are performed on different grids, the communication of data between successive grids is an important issue. We discussed two possibilities. If the usual finite element interpolation is employed, then we solve Eulerian formulation of the problem, although on a different grid at each timestep. The alternative is the standard arbitrary Lagrangian-Eulerian formulation. Numerical verification showed that the performance of the two approaches is similar.

We described in detail the implementation of the alignment algorithm in

two dimensions. The proposed procedure uses the projections of points onto the interface. Our experience has shown that projecting points directly onto the piecewise linear approximation of the interface embedded into the aligned grid does not produce satisfactory results. Instead, we construct an approximating circle in the vicinity of every interface node. The same circles are also used for the computation of the interface's curvature and normal vector, which are needed for the computation of surface tension. The proposed algorithm was validated with a series of numerical tests.

The extension of the alignment algorithm to three dimensions was met with unforeseen difficulties. A special recursive algorithm was designed to select which nodes can be projected onto the interface without introducing elements with four interface nodes. Moreover, the quality of the elements after grid alignment in three dimensions often turned out to be intolerably low, and a provision for improving the quality of the grid was added to the algorithm. Similarly to the two-dimensional case, we used spheres to approximate the interface locally. The numerical validation supported our expectations, but was less extensive due to the limitations in the available computing resources and time. Therefore, further testing should be done after proper parallelization of the code; however, such extension is beyond the scope of this work.

Conclusion

In this study, we set our goals to devise a numerical algorithm that can be used for large scale simulations of flows of multiple fluids that do not mix and possibly are subject to surface tension. The presented solution is a projection method based on a combination of conforming and nonconforming linear finite elements for the velocity. The conforming element is used to account for the convective and viscous effects, while the nonconforming one is utilized to impose the incompressibility constraint. The method also features a discontinuous approximation of the pressure, which gives two advantages. First, it makes the solenoidal velocity locally mass conserving, which facilitates stability of the advection of the free interfaces. Second, it achieves full order of interpolation, since the exact pressure is discontinuous when surface tension is present. In order to benefit from the latter point, we align the grid with the positions of the capillary surfaces at each timestep. The suggested alignment procedure is robust and computationally inexpensive. Furthermore, it guarantees that the connectivity of the computational mesh remains unchanged, which allows for a more effective parallelization.

A few challenges still remain and set new goals for future investigation. We have proved that the proposed projection scheme is optimally convergent on special grids, while numerical experiments show that the scheme is optimally convergent even on grids where our theory does not apply. The proof of convergence on general grids remains an open problem. Another open question is whether it is always possible to align the grid to the interface in three dimensions without introducing elements with four nodes on the interface.

Another issue to address in the near future is the implementation of the divergence-free projection in three dimensions. We have seen that the solenoidal basis is fully coupled and that separating the tangential components from the normal ones improves computational time significantly, while the accuracy of the solution does not deteriorate. The latter result, however, is supported only by numerical evidence at this time. We also have seen that the decoupled projection is not adequate for problems with surface tension. One may wonder whether it is possible to modify the decoupled projection so that it would become suitable for all problems, yet remain much faster than the full projection. Another possibility to research is the construction of a preconditioner that is more effective than the simple ones discussed here.

The proposed methodology is suitable for large scale simulations. The natural extension of the current study is to parallelize the code and perform simulations involving many bubbles. Afterwards, there is a multitude of possibilities for further investigation of this technique. During the alignment of the grid, we detect when topology changes are about to take place. Their appro-

priate handling would allow us to simulate such phenomena as pinch-off and coalescence. Another possibility is to incorporate a model for contact angles, which would remove the requirement for the contact surface to be closed. Yet another possible extension is to apply a different model at the interface; for example, making the interface an elastic boundary would allow us to simulate the flow of blood cells in the heart or in blood vessels.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data structures and algorithms*. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley Publishing Co., Reading, Mass., 1983.

[2] I. Babuška. Error-bounds for finite element method. *Numer. Math.*, 16:322–333, 1971.

[3] I. Babuška. The finite element method with Lagrangian multipliers. *Numer. Math.*, 20:179–192, 1973.

[4] C. Berge. *Graphs and hypergraphs*. North-Holland Publishing Co., Amsterdam, 1973. Translated from French by Edward Minieka, North-Holland Mathematical Library, Vol. 6.

[5] J.U. Brackbill, D.B. Kothe, and C. Zemach. A continuum method for modeling surface tension. *J. Comput. Phys.*, 100(2):335–354, 1992.

[6] F. Brezzi. On the existence, uniqueness and approximation of saddle-point problems arising from Lagrange multipliers. *RAIRO, Anal. Numér. R2*, pages 129–151, 1974.

[7] F. Brezzi and M. Fortin. *Mixed and hybrid finite element methods*. Springer-Verlag, New York, 1991.

[8] A. Caboussat, V. Maronnier, M. Picasso, and J. Rappaz. Numerical simulation of three dimensional free surface flows with bubbles. In *Challenges in scientific computing—CISC 2002*, volume 35 of *Lect. Notes Comput. Sci. Eng.*, pages 69–86. Springer, Berlin, 2003.

[9] T. Chen, P.D. Minev, and K. Nandakumar. A projection scheme for incompressible miltiphase flow using adaptive Eulerian grid. *Int. J. Numer. Meth. Fluids*, 45(1):1–19, 2004.

[10] T. Chen, P.D. Minev, and K. Nandakumar. A projection scheme for incompressible multiphase flow using adaptive Eulerian grids: 3d validation. *Int. J. Numer. Meth. Fluids*, 48(2):455–466, 2005.

[11] B.Y. Choi and M. Bussmann. A piecewise linear approach to volume tracking a triple point. *Internat. J. Numer. Methods Fluids*, 53(6):1005–1018, 2007.

[12] A.J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22:745–762, 1968.

[13] B. Cockburn, G. Kanshat, and D. Schoetzau. A locally conservative LDG method for the incompressible Navier-Stokes equations. *Math. Comp.*, 74:1067–1095, 2004.

[14] M. Crouzeix. In *Proc. of Journees Elements Finis.* Universite de Rennes, France, 1976.

[15] M. Crouzeix and P.-A. Raviart. Conforming and nonconforming finite element methods for solving the stationnary Stokes equations. *RAIRO, Anal. Numér.*, 3:33–75, 1973.

[16] S.J. Cummins, M.M. Francois, and D.B. Kothe. Estimating curvature from volume fractions. *Computers and Structures*, 83(6-7):425–434, 2005.

[17] C. Cuvelier, A. Segal, and A.A. van Steenhoven. *Finite element methods and the Navier-Stokes equations.* D. Reidel, 1986.

[18] M. Dai and D.P. Schmidt. Adaptive tetrahedral meshing in free-surface flow. *J. Comput. Phys.*, 208(1):228–252, 2005.

[19] J. Donea, A. Huerta, J.-Ph. Ponthot, and A. Rodríguez-Ferran. Arbitrary Lagrangian-Eulerian methods. In Erwin Stein, René de Borst, and Thomas J. R. Hughes, editors, *Encyclopedia of computational mechanics. Vol. 1 Fundamentals*, chapter 14. John Wiley & Sons Ltd., Chichester, 2004.

[20] J. Du, B. Fix, J. Glimm, X. Jia, X. Li, Y. Li, and L. Wu. A simple package for front tracking. *J. Comput. Phys.*, 213(2):613–628, 2006.

[21] F. Duarte, R. Gormaz, and S. Natesan. Arbitrary Lagrangian-Eulerian method for Navier-Stokes equations with moving boundaries. *Comput. Methods Appl. Mech. Engrg.*, 193(45-47):4819–4836, 2004.

[22] W. E and J.G. Liu. Projection method I: Convergence and numerical boundary layers. *SIAM J. Numer. Anal.*, 32:1017–1057, 1995.

[23] W. E and J.G. Liu. Gauge method for viscous incompressible flows. *Commun. Math. Sci.*, 1(2):317–332, 2003.

[24] D.A. Field. Qualitative measures for initial meshes. *Internat. J. Numer. Methods Engrg.*, 47(4):887–906, 2000.

[25] M.M. Francois, S.J. Cummins, E.D. Dendy, D.B. Kothe, J.M. Sicilian, and M.W. Williams. A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework. *J. Comput. Phys.*, 213(1):141–173, 2006.

[26] C.S. Frederiksen and A.M. Watts. Finite-element method for time-dependent incompressible free surface flow. *J. Comput. Phys.*, 39(2):282–304, 1981.

[27] L.A. Freitag and P.M. Knupp. Tetrahedral mesh improvement via optimization of the element condition number. *Internat. J. Numer. Methods Engrg.*, 53(6):1377–1391, 2002.

[28] L.A. Freitag and P. Plassmann. Local optimization-based simplicial mesh untangling and improvement. *Internat. J. Numer. Methods Engrg.*, 49:109–125, 2000.

[29] U. Ghia, K.N. Ghia, and C.T. Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *J. Comput. Phys.*, 48(1):387–411, 1982.

[30] J. Glimm, J.W. Grove, X.L. Li, and D.C. Tan. Robust computational algorithms for dynamic interface tracking in three dimensions. *SIAM Journal of Scientific Computing*, 21(6):2240–2256, 2000.

[31] J. Glimm, L.I. Xiaolin, Y. Liu, X.U. Zhiliang, and N. Zhao. Conservative front tracking with improved accuracy. *SIAM Journal on Numerical Analysis*, 41(5):1926–1947, 2003.

[32] K. Goda. A multistep technique with implicit difference schemes for calculating two- or three-dimensional cavity flows. *J. Comput. Phys.*, 30:76–95, 1979.

[33] J.-L. Guermond. Some practical implementations of projection methods for Navier-Stokes equations. *Modél. Math. Anal. Num.*, 30:637–667, 1996. Also in *C. R. Acad. Sci. Paris, Série I*, 319:887–892, 1994.

[34] J.-L. Guermond. Un résultat de convergence d'ordre deux en temps pour l'approximation des équations de Navier-Stokes par une technique de projection incrémentale. *M2AN Math. Model. Numer. Anal.*, 33(1):169–189, 1999. Also in *C. R. Acad. Sci. Paris, Série I*, 325:1329–1332, 1997.

[35] J.-L. Guermond, P. Minev, and J. Shen. An overview of projection methods for incompressible flows. *Comput. Methods Appl. Mech. Engrg.*, 195:6011–6045, 2006.

[36] J.-L. Guermond and L. Quartapelle. On the approximation of the unsteady Navier–Stokes equations by finite element projection methods. *Numer. Math.*, 80(5):207–238, 1998.

[37] J.-L. Guermond and J. Shen. Quelques résultats nouveaux sur les méthodes de projection. *C. R. Acad. Sci. Paris, Série I*, 333:1111–1116, 2001.

[38] J.-L. Guermond and J. Shen. A new class of truly consistent splitting schemes for incompressible flows. *J. Comput. Phys.*, 192:262–276, 2003.

[39] J.-L. Guermond and J. Shen. Velocity-correction projection methods for incompressible flows. *SIAM J. Numer. Anal.*, 41(1):112–134, 2003.

[40] J.-L. Guermond and J. Shen. On the error estimates for the rotational pressure-correction projection methods. *Math. Comp.*, 73(248):1719–1737, 2004.

[41] F. Hecht. Construction d'une base de fonctions P_1 non conforme á divergence nulle dans IR^3. *RAIRO Anal. Num.*, 15:119–150, 1981.

[42] C.W. Hirt, A.A. Amsden, and J.L. Cook. An arbitrary Lagrangian-Eulerian computing method for all flow speeds [J. Comput. Phys. **14** (1974), no. 3, 227–253]. *J. Comput. Phys.*, 135(2):198–216, 1997.

[43] C.W. Hirt and B.D. Nichols. Volume of fluid (VOF) method for the dynamics of free boundaries. *J. Comp. Phys.*, 39(1):201–225, 1981.

[44] J.G. Hnat and J.D. Buckmaster. Spherical cap bubbles and skirt formation. *Physics of Fluids*, 19(2):182–194, 1976.

[45] P. Hood and C. Taylor. Navier-Stokes equations using mixed interpolation. In *Proc. Finite Element Method in Flow Problems*, Swinsea, Wales, 1974. University of Alabama Press, Alabama, USA.

[46] G.E. Karniadakis, M. Israeli, and S.A. Orszag. High-order splitting methods for the incompressible Navier-Stokes equations. *J. Comp. Phys.*, 97:414–443, 1991.

[47] J. Kim and P. Moin. Application of a fractional-step method to incompressible Navier-Stokes equations. *J. Comput. Phys.*, 59(2):308–323, 1985.

[48] O.A. Ladyženskaja. *The mathematical theory of viscous incompressible flow.* Gordon and Breach Science Publishers, Inc., New York, New York, USA, 2nd edition, 1969.

[49] H. Lamb. *Hydrodynamics.* Dover, New York, 1932.

[50] R.J. LeVeque and Z. Li. Immersed interface methods for Stokes flow with elastic boundaries or surface tension.. *SIAM J. Sci. Comput.*, 18(3):709–735, 1997.

[51] Z. Li and M.C. Lai. The immersed interface method for the Navier-Stokes equations with singular forces. *J. Comp. Phys.*, 171:822–842, 2001.

[52] V. Maronnier, M. Picasso, and J. Rappaz. Numerical simulation of free surface flows. *J. Comput. Phys.*, 155(2):439–455, 1999.

[53] V. Maronnier, M. Picasso, and J. Rappaz. Numerical simulation of three-dimensional free surface flows. *Internat. J. Numer. Methods Fluids*, 42(7):697–716, 2003.

[54] F. Mashayek and N. Ashgriz. A hybrid finite-element–volume-of-fluid method for simulating free surface flows and interfaces. *Internat. J. Numer. Methods Fluids*, 20(12):1363–1380, 1995.

[55] P.D. Minev, T. Chen, and K. Nandakumar. A finite element technique for multifluid incompressible flow using Eulerian grids. *J. Comput. Phys.*, 187:255–273, 2003.

[56] S. Osher and R.P. Fedkiw. Level set methods: an overview and some recent results. *J. Comp. Phys.*, 169:463–502, 2001.

[57] S. Osher and J.A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulation. *J. Comp. Phys.*, 79(1):12–49, 1988.

[58] B. Perot and R. Nallapati. A moving unstructured staggered mesh method for the simulation of incompressible free-surface flows. *J. Comput. Phys.*, 184(1):192–214, 2003.

[59] C.S. Peskin. Numerical analysis of blood flow in the heart. *J. Computational Phys.*, 25(3):220–252, 1977.

[60] S.B. Pillapakkam and P. Singh. A level-set method for computing solutions to viscoelastic two-phase flow. *J. Comput. Phys.*, 174(2):552–578, 2001.

[61] J. Qin. *On the convergence of some low order mixed finite element methods for incompressible fluids.* Pennsylvania State University. PhD Thesis, 1994.

[62] S. Quan and D.P. Schmidt. A moving mesh interface tracking method for 3D incompressible two-phase flows. *J. Comput. Phys.*, 221(2):761–780, 2007.

[63] M. Quecedo and M. Pastor. Application of the level set method to the finite element solution of two-phase flows. *Internat. J. Numer. Methods Engrg.*, 50(3):645–663, 2001.

[64] R. Rannacher. On Chorin's projection method for the incompressible Navier-Stokes equations. In *The Navier-Stokes equations II—theory and numerical methods (Oberwolfach, 1991)*, volume 1530 of *Lecture Notes in Math.*, pages 167–183. Springer, Berlin, 1992.

[65] Y. Renardy and M. Renardy. PROST: A parabolic reconstruction of surface tension for the volume-of-fluid method. *J. Comput. Phys.*, 183(2):400–421, 2002.

[66] M. Rudman. Volume-tracking methods for interfacial flow calculations. *Internat. J. Numer. Methods Fluids*, 24(7):671–691, 1997.

[67] R.L. Sani, P.M. Gresho, R.L. Lee, and S.T. Chan. On the cause and cure (?) of the spurious pressures generated by certain FEM solutions of the incompressible navier-stokes equations: Parts 1 and 2. *Int. J. Numer. Meth. Fluids*, 1:17–43 for Part 1, 171–204 for Part 2, 1981.

[68] R. Scardovelli and S. Zaleski. Direct numerical simulation of free-surface and interfacial flow. In *Annual review of fluid mechanics, Vol. 31*, Annu. Rev. Fluid Mech., pages 567–603. Annual Reviews, Palo Alto, CA, 1999.

[69] R. Scheichl. Decoupling three-dimensional mixed problems using divergence-free finite elements. *SIAM J. Sci. Comput.*, 23(5):1752–1776, 2002.

[70] J.A. Sethian. Evolution, implementation, and application of level set and fast marching methods for advancing fronts. *J. Comput. Phys.*, 169(2):503–555, 2001.

[71] J.A. Sethian and P. Smereka. Level set methods for fluid interfaces. *Annual Review of Fluid Mechanics*, 35:341–372, 2003.

[72] J. Shen. On error estimates of projection methods for the Navier-Stokes equations: second-order schemes. *Math. Comp.*, 65(215):1039–1065, 1996.

[73] M. Sussman. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *J. Comput. Phys.*, 187(1):110–136, 2003.

[74] M. Sussman, E. Fatemi, P. Smereka, and S. Osher. An improved level set method for incompressible two-phase flows. *Comput. & Fluids*, 27(5-6):663–680, 1998.

[75] M. Sussman and E.G. Puckett. A coupled level set and volume-of-fluid method for computing 3D and axisymmetric incompressible two-phase flows. *J. Comput. Phys.*, 162(2):301–337, 2000.

[76] M. Sussman, P. Semerka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *J. Comp. Phys.*, 114(1):146–159, 1994.

[77] M. Sussman and P. Smereka. Axisymmetric free boundary problems. *Journal of Fluid Mechanics*, 341:269–294, 1997.

[78] M. Sussman, K.M. Smith, M.Y. Hussainiand M. Ohta, and R. Zhi-Wei. A sharp interface method for incompressible two-phase flows. *J. Comput. Phys.*, 221(2):469–505, 2007.

[79] C. Taylor and P. Hood. A numerical solution of the Navier-Stokes equations using the finite element technique. *Comput. Fluids.*, 1:73–100, 1973.

[80] R. Temam. Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires ii. *Arch. Rat. Mech. Anal.*, 33:377–385, 1969.

[81] T.E. Tezduyar. Finite element methods for flow problems with moving boundaries and interfaces. *Archives of Computational Methods in Engineering*, 8(2):83–130, 2001.

[82] T.E. Tezduyar. Interface-tracking and interface-capturing techniques for finite element computation of moving boundaries and interfaces. *Comput. Methods Appl. Mech. Engrg.*, 195(23-24):2983–3000, 2006.

[83] T.E. Tezduyar, M. Behr, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces – The deforming-spatial-domain/space-time procedure: I. The concept and the preliminary numerical tests. *Computer Methods in Applied Mechanics and Engineering*, 94(3):339–351, 1992.

[84] T.E. Tezduyar, M. Behr, S. Mittal, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces – The deforming-spatial-domain/space-time procedure. II. Computation of free-surface flows, two-liquid flows, and flows with drifting cylinders.

Computer Methods in Applied Mechanics and Engineering, 94(3):353–371, 1992.

[85] F. Thomasset. *Implementation of finite element methods for Navier-Stokes equations.* Springer Series in Computational Physics. Springer-Verlag, New York-Berlin, 1981.

[86] L.J.P. Timmermans, P.D. Minev, and F.N. Van De Vosse. An approximate projection scheme for incompressible flow using spectral elements. *Int. J. Numer. Methods Fluids*, 22:673–688, 1996.

[87] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A front-tracking method for the computations of multiphase flow. *J. Comp. Phys.*, 169:708–759, 2001.

[88] S. Turek. A comparative study of time-stepping techniques for the incompressible Navier–Stokes equations: from fully implicit non-linear schemes to semi-implicit projection methods. *Int. J. Numer. Meth. Fluids*, 22:987–1011, 1996.

[89] S.O. Unverdi and G. Tryggvason. A front-tracking method for viscous, incompressible, multi-fluid flows. *J. Comp. Phys.*, 100:25–37, 1992.

[90] J. van Kan. A second-order accurate pressure-correction scheme for viscous incompressible flow. *SIAM J. Sci. Stat. Comput.*, 7(3):870–891, 1986.

[91] J.L. Walters and J.F. Davidson. The initial motion of a gas bubble formed in an inviscid liquid, part 1. the two-dimensional bubble. *J. Fluid Mech.*, 12:409–417, 1962.